RESUMEN

Desde los inicios de la codificación de vídeo digital hasta hoy, tanto la señal de video sin comprimir de entrada al codificador como la señal de salida descomprimida del decodificador, independientemente de su resolución, uso de submuestreo en los planos de diferencia de color, etc. han tenido siempre la característica común de utilizar 8 bits para representar cada una de las muestras. De la misma manera, los estándares de codificación de vídeo imponen trabajar internamente con estos 8 bits de precisión interna al realizar operaciones con las muestras cuando aún no se han transformado al dominio de la frecuencia.

Sin embargo, el estándar H.264, en gran auge hoy en día, permite en algunos de sus perfiles orientados al mundo profesional codificar vídeo con más de 8 bits por muestra. Cuando se utilizan estos perfiles, las operaciones efectuadas sobre las muestras todavía sin transformar se realizan con la misma precisión que el número de bits del vídeo de entrada al codificador. Este aumento de precisión interna tiene el potencial de permitir unas predicciones más precisas, reduciendo el residuo a codificar y aumentando la eficiencia de codificación para una tasa binaria dada.

El objetivo de este Proyecto Fin de Carrera es estudiar, utilizando las medidas de calidad visual objetiva PSNR (*Peak Signal to Noise Ratio,* relación señal ruido de pico) y SSIM (*Structural Similarity*, similaridad estructural), el efecto sobre la eficiencia de codificación y el rendimiento al trabajar con una cadena de codificación/ descodificación H.264 de 10 bits en comparación con una cadena tradicional de 8 bits. Para ello se utiliza el codificador de código abierto x264, capaz de codificar video de 8 y 10 bits por muestra utilizando los perfiles *High, High 10, High 4:2:2 y High 4:4:4 Predictive* del estándar H.264.

Debido a la ausencia de herramientas adecuadas para calcular las medidas PSNR y SSIM de vídeo con más de 8 bits por muestra y un tipo de submuestreo de planos de diferencia de color distinto al 4:2:0, como parte de este proyecto se desarrolla también una aplicación de análisis en lenguaje de programación C capaz de calcular dichas medidas a partir de dos archivos de vídeo sin comprimir en formato YUV o Y4M.

ABSTRACT

Since the beginning of digital video compression, the uncompressed video source used as input stream to the encoder and the uncompressed decoded output stream have both used 8 bits for representing each sample, independent of resolution, chroma subsampling scheme used, etc. In the same way, video coding standards force encoders to work internally with 8 bits of internal precision when working with samples before being transformed to the frequency domain.

However, the H.264 standard allows coding video with more than 8 bits per sample in some of its professionally oriented profiles. When using these profiles, all work on samples still in the spatial domain is done with the same precision the input video has. This increase in internal precision has the potential of allowing more precise predictions, reducing the residual to be encoded, and thus increasing coding efficiency for a given bitrate.

The goal of this Project is to study, using PSNR (*Peak Signal to Noise Ratio*) and SSIM (*Structural Similarity*) objective video quality metrics, the effects on coding efficiency and performance caused by using an H.264 10 bit coding/decoding chain compared to a traditional 8 bit chain. In order to achieve this goal the open source x264 encoder is used, which allows encoding video with 8 and 10 bits per sample using the H.264 *High, High 10, High 4:2:2* and *High 4:4:4 Predictive* profiles.

Given that no proper tools exist for computing PSNR and SSIM values of video with more than 8 bits per sample and chroma subsampling schemes other than 4:2:0, an analysis application written in the C programming language is developed as part of this Project. This application is able to compute both metrics from two uncompressed video files in the YUV or Y4M format.

E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: CODIFICACIÓN DE VÍDEO

TÍTULO: ESTUDIO DE CALIDAD DE CODIFICACIÓN UTILIZANDO LOS PERFILES 10 BITS DEL ESTÁNDAR H.264

AUTOR: JAVIER CABEZAS GONZÁLEZ

TUTOR: MARTINA ECKERT **Vº Bº.**

DEPARTAMENTO: DIAC

Miembros del Tribunal Calificador:

PRESIDENTE: FERNANDO PESCADOR DEL OSO

VOCAL: MARTINA ECKERT

VOCAL SECRETARIO: ENRIQUE RENDÓN ANGULO

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Desde los inicios de la codificación de vídeo digital hasta hoy, tanto la señal de video sin comprimir de entrada al compresor como la señal de salida descomprimida del decodificador, independientemente de su resolución, uso de submuestreo en los planos de diferencia de color, etc. han tenido siempre la característica común de utilizar 8 bits para representar cada una de las muestras. De la misma manera los estándares de codificación imponen limitaciones en la precisión de las operaciones de muchos de los procesos internos del codificador, lo que provoca errores de redondeo que a su vez desembocan en una pérdida de eficiencia de codificación. Sin embargo, el estándar H.264, en gran auge hoy en día, permite en algunos de sus perfiles comprimir con más de 8 bits por muestra aumentando de la misma manera su precisión interna. Así pues, el objetivo de este proyecto fin de carrera es estudiar, utilizando las medidas objetivas de calidad visual PSNR y SSIM, el efecto sobre la calidad de imagen, la eficiencia de codificación y el rendimiento que proporciona el trabajar con una cadena de codificación/descodificación H.264 de 10 bits en comparación con una cadena tradicional de 8 bits.

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Universitaria

de

Ingeniería Técnica de Telecomunicación

PROYECTO FIN DE CARRERA

ESTUDIO DE CALIDAD DE CODIFICACIÓN UTILIZANDO LOS PERFILES 10 BITS DEL ESTÁNDAR H.264

Javier Cabezas González

Septiembre, 2012

ÍNDICE

ÍNDICE DE FIGURAS

Capítulo 1 - Introducción

1.1 Antecedentes y justificación

El estándar de codificación de vídeo H.264/MPEG-4 Part 10, también denominado AVC (*Advanced Video Coding*), fue desarrollado de forma conjunta por el *Video Coding Experts Group* (VCEG) de la Unión Internacional de Telecomunicaciones (UIT) y el *Moving Picture Experts Group* (MPEG) de la ISO/IEC. La primera versión del estándar fue lanzada en mayo de 2003, heredando muchas de las técnicas empleadas anteriormente por otros estándares de codificación de video como MPEG-2. El estándar, gracias a su capacidad de proporcionar calidad comparable a dichos estándares anteriores a una tasa binaria mucho menor, se ha convertido desde entonces en uno de los más usados en la actualidad. Está presente en video por internet, distribución de televisión, videoconferencia y mundo profesional, entre otros.

Esta primera versión contenía los perfiles *Baseline*, *Main* y *Extended*. Dichos perfiles estaban destinados a codificar video de resolución igual o menor a la resolución estándar SD para el mundo de consumo, con una precisión de 8 bits por muestra como máximo y submuestreo de planos de diferencia de color 4:2:0 (Figura 1.1).

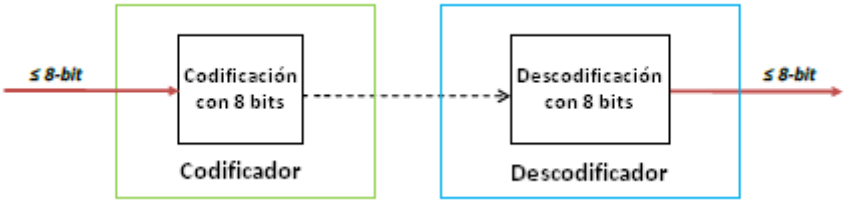

Figura 1.1 - Esquema de codificación y descodificación con 8 bits de precisión [1]

Esta versión no contemplaba el uso del estándar ni con videos de alta resolución ni en el mundo profesional, donde se requiere soporte para video con 10 bits de precisión por muestra y submuestreo de planos de diferencia de color 4:2:2 y en ocasiones 4:4:4. Por este motivo en marzo de 2005 se lanzó lo que se conoce como la extensión **FREXt** (*Fidelity Range Extensions*), la cual añadía una serie de perfiles nuevos al estándar que sí soportaban las necesidades del mundo profesional. Estos perfiles son *High*, *High 10*, *High 4:2:2* y *High 4:4:4 Predictive*. Los últimos tres permiten codificar vídeo de hasta 10 bits por muestra, y *High 4:4:4 Predictive* soporta hasta 14 bits por muestra.

Debido a la necesidad de procesar y codificar con estas nuevas profundidades de bits los codificadores que funcionan con los nuevos perfiles también deben ampliar su precisión interna a tantos bits como tenga el video que se quiera codificar. En la Figura 1.2 puede verse un diagrama de bloques de dicho proceso con una entrada de 10 bits por muestra.

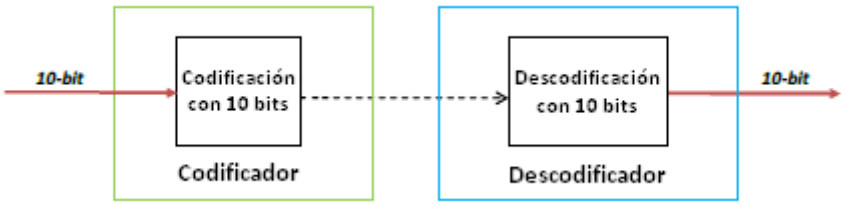

Figura 1.2 - Esquema de codificación y descodificación con 10 bits de precisión [1]

Aunque los perfiles de fidelidad extendida de más de 8 bits están diseñados para el mundo profesional donde la precisión extra es requerida, no hay nada que impida utilizar codificadores preparados para ello con video de menor número de bits por muestra, como se muestra en la Figura 1.3.

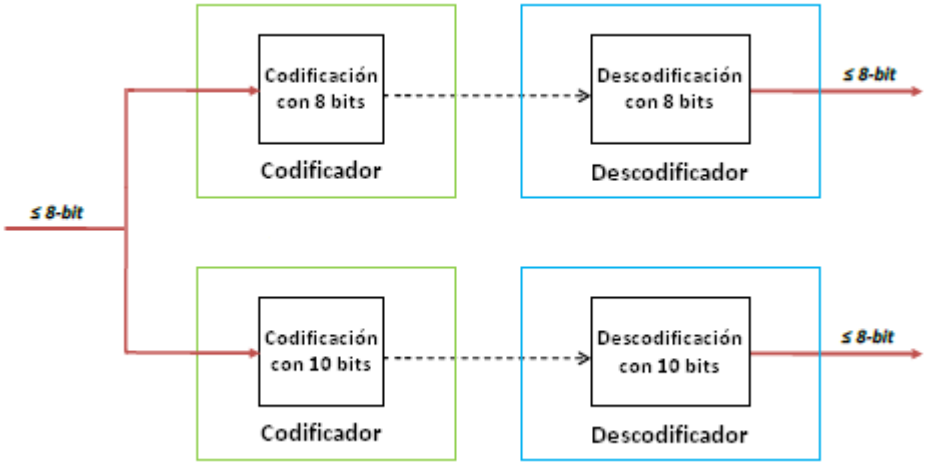

Figura 1.3 - Esquema de codificación y descodificación con 8 y 10 bits de precisión para entrada común de 8 bits [1]

Estudios previos en [2] y [3] muestran como una mayor precisión interna del codificador puede resultar en una reducción de la tasa de bits para la misma calidad de imagen.

1.2 Objetivos del proyecto

Tomando los estudios anteriormente citados como punto de partida, los objetivos de este proyecto son los siguientes:

- Estudiar, utilizando las medidas objetivas de calidad visual objetiva PSNR (*Peak Signal to Noise Ratio*) y SSIM (*Structural Similarity*), el efecto sobre la eficiencia de codificación que causa el trabajar con una cadena de codificación/descodificación H.264 de 10 bits en comparación con una cadena tradicional de 8 bits.
- Estudiar el efecto sobre el rendimiento de codificación y descodificación que produce el trabajar con los perfiles H.264 de 10 bits en comparación con los perfiles de 8 bits.
- Desarrollar un programa de análisis en lenguaje de programación C que tomando a su entrada dos archivos de video sin comprimir en formato YUV o Y4M calcule sus valores de PSNR y SSIM. Este desarrollo está motivado por la ausencia de herramientas adecuadas para calcular dichas medidas con video de más de 8 bits por muestra y tipos de submuestreo de los planos de diferencia de color distintos al 4:2:0.

1.3 Estructura de la memoria

Tras la introducción realizada en este capítulo, la memoria se completa con 7 capítulos más, que se describen a continuación.

En el Capítulo 2 se hace un repaso por las características más importantes del estándar H.264, haciendo hincapié al final en las implicaciones que tiene el aumento de precisión interna en el proceso de codificación.

En el Capítulo 3 se detallan las dos medidas de calidad visual objetiva, PSNR y SSIM, utilizadas para evaluar la calidad de video resultante de las distintas codificaciones. Además de las descripciones de ambas medidas se detalla el algoritmo de cálculo de las mismas.

En el Capítulo 4 se realiza una descripción de las distintas herramientas utilizadas durante el proyecto, tanto para el análisis de calidad como para el desarrollo de la aplicación de análisis.

En el Capítulo 5 se detalla la implementación y el funcionamiento de la aplicación de análisis diseñada por el alumno como parte del proyecto, desde la descripción de los archivos de entrada hasta la presentación de resultados.

En el Capítulo 6 se realizan las distintas pruebas de eficiencia de codificación, utilizando las herramientas descritas en los dos capítulos anteriores y las medidas de calidad objetiva PSNR y SSIM. Para ello se utilizan siete secuencias de video, conteniendo cada una diferente tipo de material.

En el Capítulo 7 se realizan las distintas pruebas de rendimiento, midiendo el impacto en la velocidad de codificación y descodificación en fotogramas por segundo de la codificación con diferente número de bits.

Por último, en el Capítulo 8 se exponen las conclusiones extraídas del proyecto y varias ideas sobre posibles trabajos futuros relacionados con lo estudiado en el mismo.

Además se han incluido dos anexos. El ANEXO I explica los diferentes parámetros que acepta la aplicación de análisis desarrollada por el alumno. En el ANEXO II se recoge una descripción de los principales parámetros de codificación que permiten configurar el codificador x264 para las distintas pruebas realizadas, así como explicar algunos de los resultados obtenidos.

Capítulo 2 - El estándar H.264

El estándar H.264 nace con el objetivo de implementar un sistema de codificación de alta compresión que sea capaz de proporcionar una calidad de imagen comparable a otros estándares de codificación de vídeo anteriores a una tasa binaria muy inferior. Fue desarrollado de forma conjunta por el *Video Coding Experts Group* (VCEG) de la Unión Internacional de Telecomunicaciones (UIT) y el *Moving Picture Experts Group* (MPEG) de la ISO/IEC.

A lo largo de este capítulo se definen las características más importantes de este estándar, de tal manera que se presupone que el lector tiene un mínimo conocimiento previo sobre codificación de vídeo en formato MPEG-2. Para una descripción más avanzada se puede consultar el estándar completo en [4].

2.1 Perfiles y niveles

En los distintos estándares de codificación se suelen definir lo que se denominan perfiles y niveles.

Los perfiles definen los conjuntos de características que deben soportar tanto codificadores como descodificadores si quieren declararse como compatibles con dicho perfil. La primera versión del estándar lanzada en mayo de 2003 contenía 3 perfiles:

- *Baseline*: destinado a dispositivos con recursos limitados y para transmisiones tipo videoconferencia donde el régimen binario es reducido.
- *Main*: destinado a dispositivos generales de consumo así como para transmisión de televisión en resoluciones estándar SD.
- *Extended*: destinado a servicios multimedia en internet. Añade características que lo hacen muy resistente frente a errores de transmisión donde son frecuentes las pérdidas de paquetes.

Más adelante, en marzo de 2005, se lanzó lo que se conoce como la extensión FREXt (*Fidelity Range Extensions*), la cual añadía una serie de perfiles nuevos al estándar. En este proyecto se trabaja con estos perfiles.

- *High*: destinado a codificar video de alta resolución HD con un máximo de 8 bits por muestra como los perfiles inferiores y únicamente con el tipo de submuestreo de planos de diferencia de color 4:2:0.
- *High 10*: idéntico al perfil *High*, pero con capacidad de codificar video de hasta 10 bits por muestra.
- *High 4:2:2*: idéntico al perfil *High 10*, pero aparte del formato de submuestreo 4:2:0 permite usar el formato 4:2:2.
- *High 4:4:4 Predictive*: con todas las características del perfil High 4:2:2, pero permite utilizar también el formato 4:4:4. Extiende el número de bits por muestra codificables hasta 14. Además añade un modo de codificación sin pérdidas *lossless*.

Los niveles definen distintos limites en parámetros de codificación como la tasa máxima de bits por segundo o el máximo número de macrobloques que los decodificadores deben ser capaces de cumplir si quieren declararse como compatibles con dicho nivel. Para una descripción completa de los 17 niveles ofrecidos consultar el estándar en [4].

2.2 Tipos de fotogramas

En H.264 existen cinco tipos posibles de fotogramas a codificar. A continuación se describen cada uno de ellos basándose en las características que les diferencia del resto de fotogramas.

Un concepto importante relacionado con anteriores estándares es el GOP (*Group Of Pictures*). Un GOP está formado por una secuencia de fotogramas consecutivos que comienza y finaliza con un fotograma tipo I. H.264 no refleja la definición de GOP. Esto se debe a que en realidad H.264 no tiene por qué usar más de un fotograma tipo I dentro de una secuencia de video. Trabajando con un solo fotograma I al inicio nunca se finalizaría el GOP, de ahí que no se utilice este concepto dentro del estándar.´

- Fotogramas tipo **I**

  Un fotograma I es un fotograma codificado sin referencia a ningún fotograma anterior. Al no utilizar referencias, el tiempo invertido en su codificación es el más bajo respecto a otros tipos de fotogramas. En cambio, la carga binaria que presenta es la más elevada dentro de la secuencia de video, ya que al no tomar como referencia otros fotogramas la información a codificar es mayor. Este tipo de fotogramas utilizan predicción *intra*, la cual se explicará más adelante. Estos

fotogramas pueden dividirse en dos tipos: fotogramas **IDR** y los tradicionales fotogramas I. Un fotograma IDR es aquel que fuerza a que ningún fotograma posterior a él en el tiempo pueda referenciar a otro fotograma anterior a él en el tiempo, mientras que fotogramas colocados a continuación de un fotograma I normal en orden temporal pueden referenciar fotogramas anteriores a él en el tiempo. La definición de fotograma tipo I en estándares anteriores coincide con la de IDR en H.264, no con la de tipo I normal. Por este motivo es necesario un fotograma IDR para iniciar la reproducción del video con garantías de no encontrarse algún fotograma corrupto por faltar alguno de sus fotogramas de referencia anteriores. También son considerados puntos de refresco, no permitiendo la propagación de errores.

- Fotogramas tipo **P**

  Los fotogramas tipo P explotan la redundancia temporal existente entre varios fotogramas próximos en el tiempo. De esta manera, un fotograma tipo P puede referenciar secciones de video similares en fotogramas anteriores y codificar sólo la diferencia entre éstas y el fotograma actual. Al enviar sólo esta diferencia la carga binaria es menor a la que tienen los fotogramas I. El tiempo de codificación se ve incrementado respecto a los fotogramas tipo I, ya que la búsqueda de fotogramas de referencia ralentiza la codificación. Este tipo de fotogramas utilizan predicción *inter*, la cual se explicará más adelante.

- Fotogramas tipo **B**

  Los fotogramas tipo B son similares a los fotogramas tipo P, explotando también la redundancia temporal. En este caso los fotogramas pueden usar como referencias fotogramas posteriores a ellos en el tiempo, en lugar de hacerlo sólo con fotogramas pasados como los tipo P. Estos fotogramas hacen disminuir en gran medida la velocidad de codificación del sistema. Pueden usarse como fotogramas de referencia fotogramas tipo I, fotogramas tipo P o incluso otros fotogramas tipo B. En estándares anteriores los fotogramas B no podían utilizarse como fotogramas de referencia. Para codificar fotogramas tipo B se requieren menos bits que para codificar fotogramas tipo I o tipo P, ya que la predicción realizada entre los dos fotogramas de referencia puede combinarse provocando que el residuo a codificar sea menor.

- Fotograma tipo **SI** y **SP**

  Estos tipos de fotogramas solo se utilizan en el perfil *Extended*, no usado en este proyecto, por lo tanto no son tratados aquí.

2.3 Macrobloques y *slices*

Un fotograma está compuesto por un conjunto de MB (macrobloques). Un MB se define como una agrupación de pixeles dentro un fotograma (16x16 pixeles), como puede verse en la Figura 2.1.

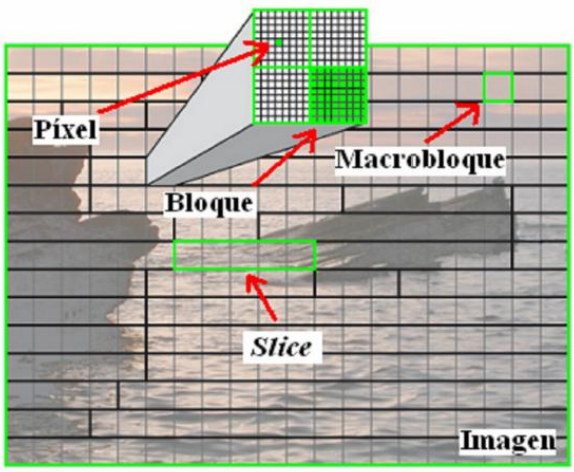

Figura 2.1 - Píxeles, bloques, macrobloques y *slices* [5]

Al igual que un fotograma puede ser de diferentes tipos, los MB también lo pueden ser. Existen cuatro tipos distintos: MB I, MB P, MB B y MB *Skip*.

Un MB tipo I se codifica sin necesidad de disponer de otros MB de referencia (predicción *intra*). En cambio, los MB tipo P y tipo B se predicen de otros MB situados en otros fotogramas dentro de la misma secuencia de video (predicción *inter*). Los MB tipo P solo buscan referencias en MB de fotogramas anteriores en el tiempo al actual y los MB tipo B buscan referencias en fotogramas tanto anteriores como posteriores en el tiempo.

El caso particular que presentan los MB son los MB *Skip*. Este tipo de MB no se codifica. Esto se debe a que el contenido del MB es igual o muy similar al de sus MB vecinos, por lo tanto el codificador entiende que no hay necesidad de codificar ambos y lo marca como tal. Esta situación se da con bastante frecuencia, debido a las grandes similitudes espaciales y temporales de los MB de una misma secuencia.

Un fotograma tipo I solo puede estar formado por MB tipo I. Un fotograma tipo P puede estar formado por MB tipo I, P o *Skip*. Por último un fotograma tipo B puede estar formado por MB tipo I, P, B o *Skip*.

Un *slice* o rebanada se define como un conjunto de MB. Cada fotograma puede tener uno o varios *slices*, cada uno de los cuales puede decodificarse independientemente de

los demás ya que no hay dependencias entre ellos. Los *slices* se utilizan para que la descodificación de los distintos *slices* presentes en un fotograma pueda realizarse en paralelo, aumentando la velocidad de dicho proceso.

2.4 Predicción *intra*

Como ya se ha comentado, un fotograma o MB tipo I no explota la redundancia temporal presente en la secuencia. Para reducir el régimen binario necesario para codificar un MB I en estándares anteriores se aplicaban una serie de técnicas como la codificación diferencial del coeficiente DC en el estándar MPEG-2. En H.264 se utiliza un sistema de predicción más complejo que explota la redundancia espacial, conocido como predicción *intra*.

Ésta consiste en, utilizando los valores de los pixeles adyacentes al MB a codificar, realizar una predicción de dicho MB. A continuación se calcula la diferencia de esta predicción con el MB real a codificar, codificándose solo dicha diferencia. De esta manera se reduce considerablemente la información a codificar explotando la redundancia espacial existente entre MB contiguos.

Existen nueve modos diferentes de predicción para luminancia (con bloques de 4x4 o de 16x16) como se ve en la Figura 2.2, y solo un modo para crominancia (para bloques de 4x4).

Un ejemplo del potencial de este tipo de predicción puede verse en la Figura 2.3, donde se tiene por un lado el fotograma a codificar y por otro los bloques de 4x4 pixeles predichos por el codificador. Únicamente la diferencia entre éstos y el original será codificada, resultado en una importante ganancia en eficiencia de codificación. Un codificador bueno seguirá el proceso de, para cada bloque, analizar todas las predicciones posibles y ver cual presenta la menor diferencia con el bloque a codificar, de tal manera que se minimice el residuo.

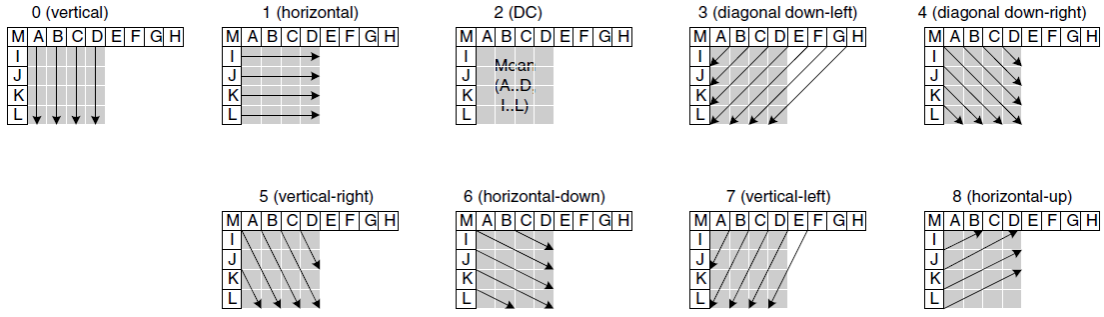

Figura 2.2 - Distintos modos de predicción *intra* en bloques 4x4 de luminancia [6]

Figura 2.3 - Fotograma original a la izquierda, predicción *intra* del mismo a la derecha [6]

2.5 Predicción *inter*

Este tipo de predicción consiste en la explotación de la redundancia temporal existente entre los distintos fotogramas que conforman una secuencia. Dicha redundancia suele ser elevada, por lo que un buen sistema de predicción *inter* puede reducir considerablemente la cantidad de información a codificar. Este tipo de predicción solo se utiliza en fotogramas tipo P y fotogramas tipo B.

Respecto a otros estándares anteriores, la predicción *inter* de H.264 presenta una serie de diferencias:

1) Mayor flexibilidad de particiones de bloques.
2) Resolución de hasta 1/4 de píxel para la estimación y compensación de movimiento.
3) Múltiples referencias, incluyendo fotogramas B.

- **Particiones**

Cada MB de luminancia puede dividirse en las siguientes particiones (Figura 2.4):

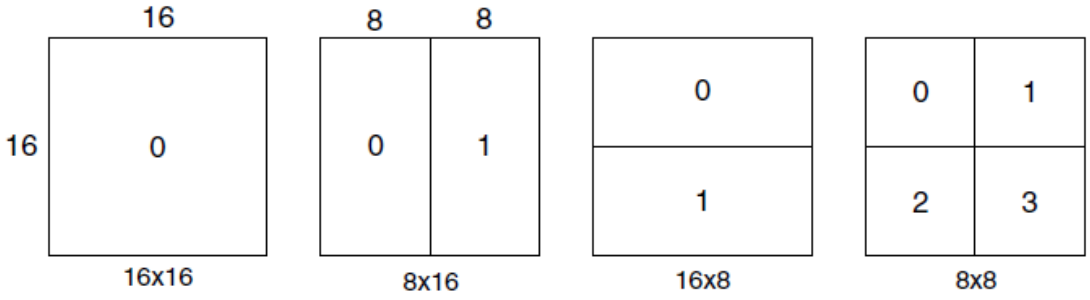

Figura 2.4 - Particiones MB: 16x16, 8x16, 16x8 y 8x8 [6]

De utilizarse la división 8x8, dichos bloques pueden a su vez dividirse en lo que se conoce como particiones sub-MB (Figura 2.5):

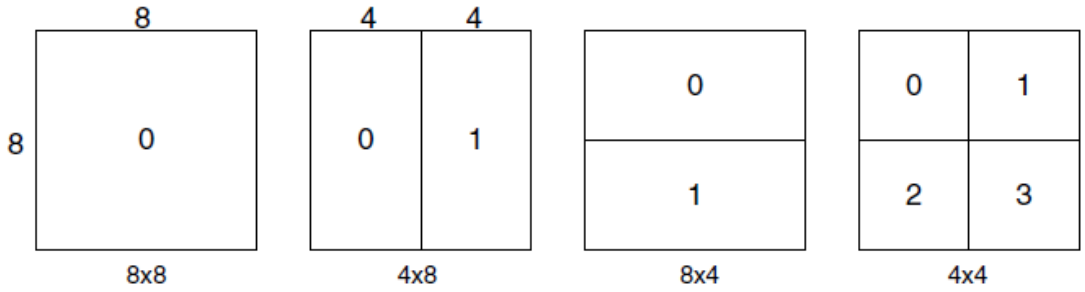

Figura 2.5 - Particiones sub-MB: 8x8, 4x8, 8x4 y 4x4 [6]

Cada partición se utiliza dependiendo del fotograma que se vaya a codificar. Las particiones más grandes se aplican a aquellos MB que son muy uniformes, sin grandes detalles. Por el contrario, una partición pequeña se usa en situaciones que requieren mayor precisión a la hora de encontrar la mejor predicción. Por lo tanto, se asignan bloques de diferente tamaño según sea la cantidad de detalle y movimiento existente entre fotogramas. Un ejemplo puede verse en la Figura 2.6, donde en zonas con mayor cantidad de detalle se usan particiones más pequeñas:

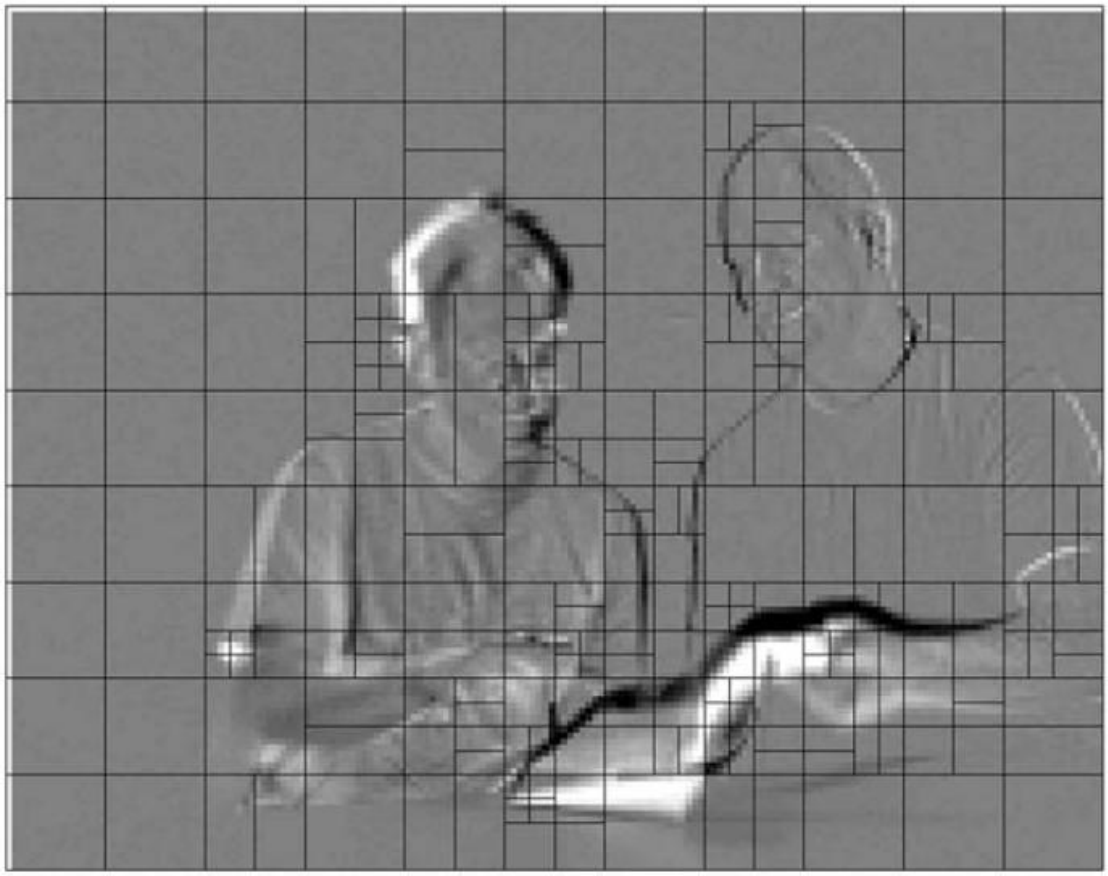

Figura 2.6 - Elección de diferentes particiones según el nivel de detalle en la etapa de compensación de movimiento [6]

De la misma manera que en el caso de la predicción *intra*, un buen codificador elegirá aquellas particiones tales que la diferencia a codificar más el coste en bits del vector de movimiento sean los mínimos posibles.

- **Estimación y compensación de movimiento**

Una vez que se ha encontrado el fotograma o fotogramas de referencia idóneos para el fotograma que se va a codificar, se comparan sus MB. La posición del MB de referencia que se haya elegido para realizar la compensación de movimiento respecto al MB a codificar queda definida por el vector de movimiento. Este vector indica cuánto se ha movido el MB que se está analizando respecto al MB tomado como referencia, dando una posición exacta tanto en horizontal como en vertical. El método que identifica los vectores de movimiento se denomina estimación de movimiento. En el caso de que ambos MB fueran exactamente iguales, solo haría falta codificar y trasmitir la información del vector de movimiento. En caso de que no fuera así aparte de

codificar y enviar el vector de movimiento hay que codificar la diferencia entre el MB a codificar y el MB de referencia. Un ejemplo del cálculo de un vector de movimiento puede verse en la Figura 2.7

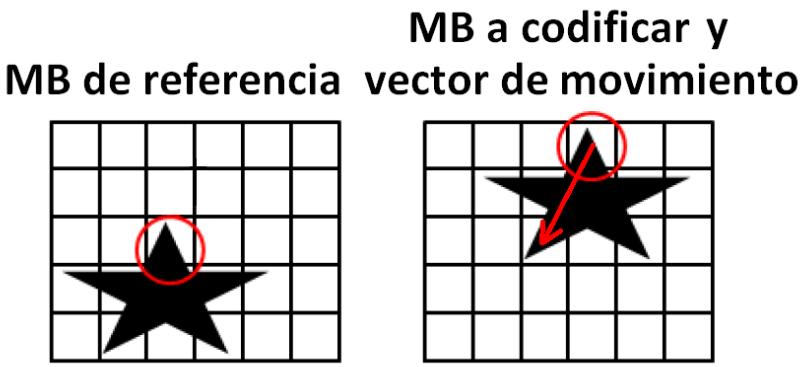

Figura 2.7 - Cálculo de un vector de movimiento [7]

- **Resolución de hasta ¼ de pixel en estimación y compensación de movimiento**

  Aunque la unidad básica de información de video es el pixel, en muchas ocasiones la secuencia a codificar presenta un movimiento tal que los objetos no se desplazan una distancia múltiplo de un pixel, como en la Figura 2.8.

  Por ello la estimación de movimiento se puede realizar mediante muestras enteras (pixeles), o precisiones mayores según sean requeridas. Esto provoca que haya que interpolar y refinar el área que se está utilizando como referencia para crear esos valores intermedios que luego se utilizan como referencia.

  En estándares anteriores se usaban muestras de ½ pixel, pero en H.264 se puede trabajar con muestras de hasta ¼ de pixel, aumentando la precisión de su bloque estimación de movimiento y por lo tanto realizando predicciones más precisas.

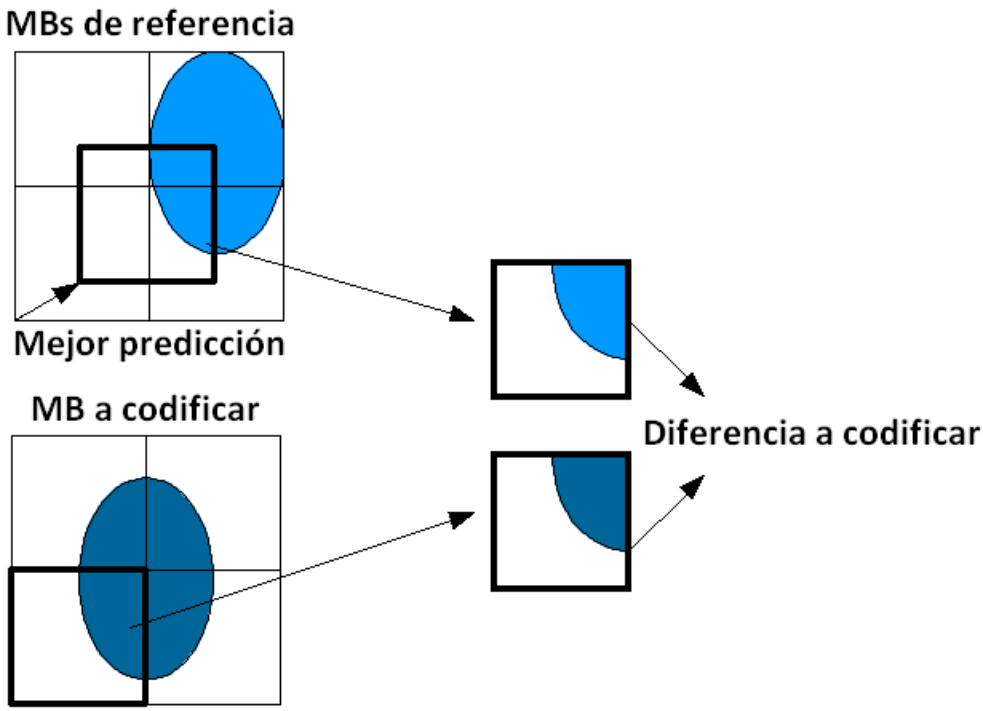

Figura 2.8 - Utilidad de la precisión sub-pixel [8]

- **Múltiples referencias**

  En H.264 se definen dos listas que almacenan los fotogramas que se van a usar como fotogramas de referencia para otros fotogramas. La lista 0 se encarga de los fotogramas pasados, aquellos que ya han sido codificados, y la lista 1 almacena los fotogramas que se van a codificar en un futuro. Los fotogramas tipo P solo buscan referencias en la lista 0, en cambio los fotogramas B buscan en ambas.

  En estándares anteriores la predicción solo se permitía con un solo fotograma aunque se tuvieran varios fotogramas de referencia donde buscar y elegir. En cambio en H.264 existe la opción de usar múltiples fotogramas como referencia. El resultado de esta predicción se hace mediante la suma ponderada de los MB de los fotogramas de referencia elegidos, para formar una predicción lo más cercana posible al MB a codificar (Figura 2.9).

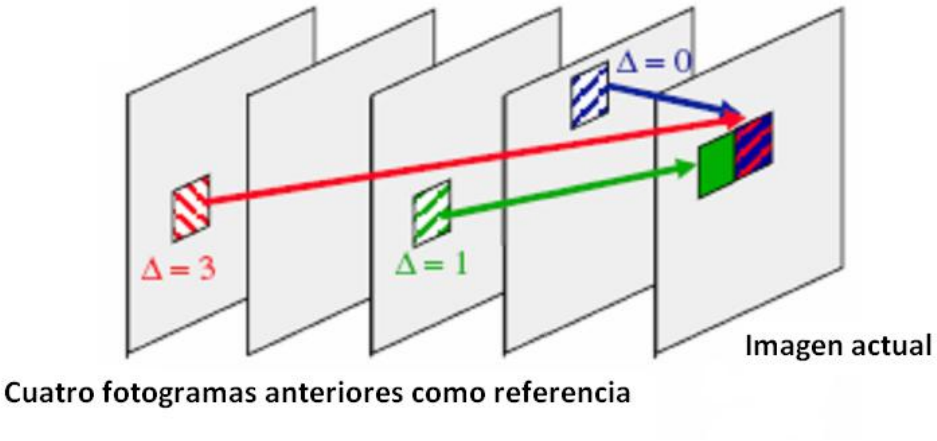

Figura 2.9 - Uso de múltiples fotogramas como referencia [5]

2.6 Filtro *in-loop deblocking*

El filtro antibloques en lazo (*in-loop deblocking filter*) se encarga de suavizar los bordes de los bloques de los fotogramas que aparecen después de una codificación utilizando transformadas por bloques cuadrados de pixeles. Dicho efecto es muy desagradable visualmente, y en el propio proceso de codificación tiene consecuencias nefastas a la hora de utilizar un fotograma con el mencionado efecto como referencia de otros fotogramas, tanto en la calidad como en la eficiencia de codificación. Con la intención de atajar este problema se introduce un filtro adaptativo antibloques para paliar este efecto dentro del bucle de compensación de movimiento.

Para cada bloque se comprueba si en la unión de éste con los bloques adyacentes hay una diferencia de contraste considerable y si es así se suavizan los bordes de dichos bloques mediante el filtrado. Se trata de un filtro paso bajo que elimina las fuertes variaciones presentes cuando se produce el efecto y deja como resultado un degradado entre un bloque y otro donde antes había un salto brusco. En la Figura 2.10 puede observarse un ejemplo de la acción de este filtro.

Figura 2.10 - Fotograma sin filtrar a la izquierda, mismo fotograma filtrado con filtro antibloques a la derecha [6]

2.7 Transformada y cuantificación

Una vez se han realizado las predicciones, se tiene como resultado un residuo o error. Estos residuos deben dividirse en bloques de 4x4 o en bloques de 8x8 para poder llevar a cabo su codificación. Se convierten al dominio de la frecuencia y se cuantifican.

Para pasarlos al dominio de la frecuencia se utiliza la transformada de enteros (*Integer Transform*) que es una aproximación de la DCT (*Discrete Cosine Transform*, Transformada Discreta del Coseno), puesto que tiene características similares a la DCT, pero usando coeficientes enteros. Existen dos tamaños de transformada, 4x4 y 8x8 (en vez de solo 8x8 en la DCT), elegibles por el codificador según cual de ellas considere que es adecuada en cada momento.

Una característica muy importante de esta transformada es que puede ser realizada utilizando solo sumas y desplazamientos, incrementando drásticamente el rendimiento respecto a la DCT tradicional. Además los resultados pueden ser obtenidos utilizando sólo operaciones aritméticas de 16 bits de precisión, en lugar de los 32 bits o más requeridos en la DCT usada en anteriores estándares [9], lo cual incrementa aun más el rendimiento de la transformada.

Otra ventaja de esta transformada es que debido a la utilización de coeficientes enteros en lugar de usar coeficientes fraccionales al realizar la transformada inversa se obtienen exactamente los mismos valores que se tenían antes de realizar la transformación. Esto consigue evitar los errores de redondeo tan habituales al usar la DCT tradicional y todos los errores derivados de dicho redondeo que sufría por ejemplo MPEG-2.

Una vez obtenidos los distintos coeficientes tras la transformada éstos se cuantifican según la tasa binaria que se quiera obtener, teniendo en cuenta que cuanto más se cuantifique más calidad se perderá en la codificación.

2.8 Reordenación de coeficientes y codificación entrópica

Una vez obtenidas las matrices de coeficientes hay que serializarlos para que se pueda realizar la codificación entrópica. El método utilizado es el mismo que en MPEG-2, zig-zag. Consiste en ir recorriendo la matriz de coeficientes de la forma que indica la Figura 2.11.

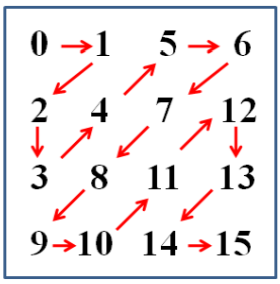

Figura 2.11 - Recorrido en zig-zag de ordenación de coeficientes [7]

A continuación, se lleva a cabo la codificación entrópica, para reducir al máximo el régimen binario sin provocar pérdidas. Existen dos métodos principales:

- CAVLC (*Context Adaptive Variable Length Coding*)

  En este modo los coeficientes resultantes de la cuantificación se codifican utilizando tablas VLC (*Variable Length Coding*) que van variando en el tiempo en función de los datos que se van enviando, asignando menos bits a los valores que se van viendo menos probables. Es el método más rápido de los dos, pero también el que menos comprime.

- CABAC (*Context Adaptive Binary Arithmetic Coding*)

  En este modo en lugar de utilizar tablas VLC se utiliza codificación aritmética. Con este método se hace un estudio más profundo de las probabilidades de los diferentes códigos y como resultado, para la misma secuencia, la tasa de bits puede llegar reducirse entre un 10% y un 20% respecto a CAVLC.

2.9 Cambios en los perfiles con más de 8 bits de precisión

Como se ha visto a lo largo del capítulo según se han ido definiendo las características del estándar, la principal herramienta que tiene un codificador de video para reducir la tasa binaria dedicada a codificar cada uno de los MB son los diferentes tipos de predicciones. Una mejor predicción implica una menor diferencia con el MB a codificar, y por tanto hacen falta menos bits para codificar esa diferencia. Por ello cualquier mejora en esas predicciones resulta en un aumento de la calidad para una misma tasa binaria, o una menor tasa binaria para obtener la misma calidad. Y es precisamente en esas mejoras en las predicciones donde un aumento de precisión interna es útil, ya que se eliminan errores de redondeo que permiten predicciones más precisas.

Como ya se ha explicado los perfiles superiores del estándar permiten codificar video con más de 8 bits por muestra. De la misma manera que un codificador de video de 8 bits de precisión almacena los pixels y trabaja internamente con esos 8 bits en todos los bloques donde todavía no se ha realizado la transformada al dominio de la frecuencia (predicción *intra*, estimación y compensación del movimiento y filtro *in-loop deblocking*), un codificador que codifique video con 10 o 14 bits por muestra trabaja internamente con 10 o 14 bits en esos mismos bloques que el codificador a 8 bits.

Debido a esto, si se convierte video que en origen es de 8 bits a 10 bits o más poniendo sus bits menos significativos a cero y se utiliza el codificador en modo de 10 bits o más, se están utilizando esos bits de precisión extra, que es precisamente lo que se pretende estudiar en este proyecto.

En el caso anterior se produce un aumento de precisión en todos aquellos pasos del proceso de codificación que ocurren en el dominio espacial: estimación y compensación del movimiento, predicción *intra* y filtro *in-loop debloking*. Mayor precisión implica una potencial mejora en la eficiencia de codificación al poderse hacer predicciones más precisas, lo que trae consigo residuos a codificar de menor tamaño que necesitan menos bits para ser codificados. En capítulos posteriores se estudia el efecto que esta precisión extra tiene al codificar diferentes secuencias de video.

Respecto al paso tanto de transformación en codificación como de transformación inversa en descodificación, el número de bits necesarios para realizar las operaciones es mayor en el caso de codificar muestras de más de 8 bits. Como se ha comentado en el apartado de transformación, la versión inicial del estándar fue creada con el objetivo de que dicha transformación pudiera ser realizada utilizando solo sumas y desplazamientos para que fuese rápida de calcular, y que además dichas operaciones pudieran ser realizadas utilizando sólo 16 bits de precisión (el valor máximo tras la

transformación no excede ±32768). Al trabajar con muestras de más bits, y por tanto con un mayor rango dinámico, 16 bits no son suficientes y hay que utilizar 32 bits.

En la Figura 2.12 se muestra una parte del código fuente del codificador x264. Dicha parte fue añadida cuando en julio de 2010 se introdujo soporte para los perfiles de más de 8 bits. Las modificaciones completas pueden consultarse en [10].

```
#if HIGH_BIT_DEPTH
    typedef uint16_t pixel;
    typedef uint64_t pixel4;
    typedef int32_t  dctcoef;
    typedef uint32_t udctcoef;

#   define PIXEL_SPLAT_X4(x) ((x)*0x0001000100010001ULL)
#   define MPIXEL_X4(src) M64(src)
#else
    typedef uint8_t  pixel;
    typedef uint32_t pixel4;
    typedef int16_t  dctcoef;
    typedef uint16_t udctcoef;

#   define PIXEL_SPLAT_X4(x) ((x)*0x01010101U)
#   define MPIXEL_X4(src) M32(src)
#endif
```

Figura 2.12 - Diferentes tipos de datos en el codificador x264 según número de bits a codificar [10]

Dicho código concuerda perfectamente con lo expuesto anteriormente. La variable HIGH_BIT_DEPTH toma valor 1 cuando el codificador se configura para ser compilado con soporte de más de 8 bits, y 0 en caso contrario. En efecto, el tipo de variable utilizada para todas aquellas operaciones que trabajen con los valores de los pixeles sin transformar al dominio de la frecuencia se incrementa de 8 a 16 bits sin signo.

Además, el tipo de variable utilizada para los coeficientes una vez transformados al dominio de la frecuencia pasa de 16 bits a 32 bits.

Aunque el aumento de precisión interna resulta en una mejora de las predicciones, el trabajar con variables de mayor tamaño tiene un efecto negativo sobre el rendimiento. Dicho efecto se estudia en el Capítulo 7.

Capítulo 3 - Medidas de calidad objetiva utilizadas en las pruebas

En este capítulo se explican las medidas de calidad objetiva utilizadas en el proyecto para medir las diferencias existentes entre la codificación con 8 y 10 bits de precisión interna. Estas medidas son PSNR (*Peak Signal to Noise Ratio*, relación señal ruido de pico) y SSIM (*Structural Similarity*, similaridad estructural).

3.1 PSNR

El PSNR es una de las medidas de calidad visual objetiva más utilizadas desde los comienzos de la codificación de video digital. Define la relación entre la máxima energía posible de una señal y el ruido que afecta a su representación fidedigna. Dado el gran rango dinámico de las señales de vídeo, donde el valor de cada muestra puede ir desde 0 hasta 255 para videos de 8 bits por muestra y desde 0 hasta 1023 para videos de 10 bits por muestra, su valor se suele representar en escala logarítmica, con el decibelio (dB) como su unidad. El PSNR se calcula mediante la siguiente expresión:

$$PSNR = 10\log\left(\frac{MAX^2}{MSE}\right) = 20\log\left(\frac{MAX}{\sqrt{MSE}}\right) \quad (1)$$

Donde el MSE (*Mean Squared Error*, error cuadrático medio) de las dos imágenes a comparar se calcula mediante la expresión:

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}|O(i,j) - D(i,j)|^2 \quad (2)$$

En la cual **M** y **N** son el ancho y el alto de la imagen en pixeles, respectivamente, y donde **O(i, j)** y **D(i, j)** son cada pixel situado en la fila **i** y la columna **j** de la imagen original y distorsionada, respectivamente.

MAX representa el valor máximo que puede tomar cada pixel. Este valor depende del número de bits por muestra de la señal, y como en este proyecto se trabaja con señales que poseen diferente número de bits por muestra **B**, se define como:

$$MAX = 2^B - 1, \tag{3}$$

lo cual equivale a 255 para señales de 8 bits por muestra, y 1023 para señales de 10 bits por muestra.

Esta medida está considerada como no realista [11], en el sentido de que no presenta correlación con el sistema visual humano. No obstante el PSNR es una medida de calidad útil si las distintas secuencias codificadas a comparar están todas hechas con parámetros de codificación que maximicen su PSNR para una tasa de bits dada, ya que se da por hecho que en este caso un PSNR más alto significa una mayor calidad de imagen subjetiva posterior.

Debido a esto el análisis de calidad de este proyecto se efectúa maximizando siempre el PSNR para cada régimen binario dado.

3.2 SSIM

Esta medida de calidad visual objetiva, creada por Zhou Wang en 2004 [12], nace con la intención de crear una medida objetiva más cercana al sistema visual humano que el PSNR.

La manera más sencilla de entender esta medida es la siguiente: mientras que el PSNR es una medida basada en el error entre dos señales, SSIM es una medida de distorsión estructural. Para su desarrollo el autor se basó en el hecho de que una de las funciones principales del ojo humano es extraer información estructural del campo de visión, ya que los humanos tienen una tendencia muy desarrollada a visualizar formas y contornos de manera precisa. SSIM considera la degradación de la imagen como pérdida de la información estructural percibida. En la Figura 3.1 se puede ver un ejemplo en el cual para valores de MSE (y por tanto, de PSNR) similares, la calidad de imagen subjetiva varía enormemente, estando SSIM mucho más cercano a la calidad subjetiva percibida por los humanos. Aun así, no debe olvidarse que SSIM sigue siendo una medida puramente matemática que nunca podrá llegar a representar de manera fiel al sistema visual humano.

Figura 3.1 - Imagen de Lena con distintas distorsiones [13]

- Arriba a la izquierda: imagen con cambio de media, MSE = 225, SSIM = 0,9894

- Arriba a la derecha: imagen con extensión de contraste, MSE = 225, SSIM = 0,9372

- Abajo a la izquierda: imagen desenfocada, MSE = 225, SSIM = 0,3461

- Abajo a la derecha: imagen comprimida JPEG, MSE = 215, SSIM = 0,2876

El cálculo del SSIM de una imagen se realiza en ventanas de pequeño tamaño, normalmente de 8x8 pixeles, mediante la siguiente expresión:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (4)$$

Donde **x** e **y** son las ventanas original y distorsionada respectivamente del mismo tamaño NxN, $\boldsymbol{\mu_x}$ la media de **x**, $\boldsymbol{\mu_y}$ la media de **y**, $\boldsymbol{\sigma_x^2}$ la varianza de **x**, $\boldsymbol{\sigma_y^2}$ la varianza de

y, σ_{xy} la covarianza de **x** e **y**, y c_1 y c_2 dos constantes para estabilizar la división en caso de que el denominador tienda a cero, con los valores:

$$c_1 = (0{,}01 \cdot MAX)^2 \text{ y}$$
$$c_2 = (0{,}03 \cdot MAX)^2, \quad (5)$$

siendo **MAX** el valor máximo que puede tomar cada pixel, 255 para señales de 8 bits por muestra y 1023 para señales de 10 bits por muestra. Una vez realizado el cálculo de *n* ventanas, cuyo número dependerá de la resolución de la imagen y del algoritmo utilizado para el cálculo, el valor final de SSIM es la media aritmética de los valores de todas las ventanas.

Los valores de SSIM pueden ir desde 0 hasta 1. Cuanto más se acerque el valor a 1 menos distorsión estructural habrá entre las dos imágenes, siendo el valor 1 solo obtenible en el caso de dos imágenes exactamente iguales. SSIM suele usarse solo con los planos de luminancia de cada fotograma, debido a la poca información estructural contenida en los planos de diferencia de color.

Como en el caso del PSNR, el análisis de calidad de este proyecto se efectúa maximizando siempre el SSIM para cada régimen binario dado.

Ya que el cálculo de SSIM de una imagen se realiza por la aproximación de ventanas, conviene resaltar la elección de dichas ventanas en este proyecto. El autor original no especifica un criterio de elección de número o posición de las distintas ventanas, por lo que aplicación de análisis de calidad desarrollada como parte de este proyecto utiliza el siguiente algoritmo, explicado en pseudocódigo:

```
i, j, número de ventanas, SSIM total;
i = j = número de ventanas = SSIM total = 0;
Mientras que i < altura - 8
{
      Mientras que j < anchura - 8
      {
            Calcular SSIM ventana 8x8;
            SSIM total = SSIM total + SSIM ventana;
            número de ventanas = número de ventanas + 1;
            j = j + 4;
      }
      i = i + 4;
}
SSIM total = SSIM total / número de ventanas;
```

Lo cual equivale a crear una malla imaginaria de 4x4 pixeles sobre la imagen a analizar, y colocar cada ventana de 8x8 pixeles con el pixel superior izquierdo en las intersecciones de dicha malla. Este algoritmo, sacado de [14], hace que las ventanas se

solapen entre sí y que una de cada dos no caiga justo en los bordes de los bloques del video codificado, lo cual penaliza el efecto de *blocking* si existe.

Capítulo 4 - Entorno de trabajo

En este capítulo se describen las aplicaciones y servicios utilizados para el desarrollo del proyecto. Son las siguientes:

- x264: para codificar video según el estándar H.264
- FFmpeg: para decodificar archivos codificados siguiendo el estándar H.264 a archivos sin comprimir *.yuv* o *.y4m*.
- Microsoft Visual Studio 2010: para el desarrollo de la aplicación de análisis de calidad objetiva.
- Google Code: para actuar como *hosting* del sistema de control de versiones Subversion [15] utilizado para manejar el código de la aplicación de análisis.
- TortoiseSVN: para actuar como cliente Subversion y acceder al repositorio desde el ordenador.
- GraphStudioNext y LAV Filters: para la medida del rendimiento de descodificación.

A continuación se procede a describir dichas herramientas con más detalle.

4.1 El codificador x264

x264 es una librería de software libre licenciada bajo licencia GNU GPL que permite la codificación de secuencias de vídeo en formato H.264. Los autores, aparte de desarrollar la librería de codificación en sí, proveen de una interfaz de línea de comandos que funciona en numerosos sistemas operativos, entre los que están los más utilizados: Windows y Linux. El desarrollo de esta librería es un proceso constante que a fecha de hoy todavía no ha terminado, ya que se siguen aportando parches de código que corrigen errores o añaden nuevas funcionalidades. Dicho código puede consultarse y descargarse en [16]. Debido al su desarrollo constante y a la no existencia de versiones oficiales, los usuarios siempre deben descargar el ejecutable compilado a partir de la última revisión del código disponible de la página oficial [17]. En el momento de la realización de este proyecto la última revisión disponible es la **r2208** de la aplicación x264 en línea de comandos para Windows, y por lo tanto esa es la que se utiliza durante el desarrollo del mismo. Como los ejecutables de la página oficial son

sustituidos por revisiones más nuevas con frecuencia, se ofrece un link al código concreto de la revisión 2208 en [18].

A continuación se muestran algunas de las características más relevantes del estándar H.264 que x264 soporta:

- B-pyramid (fotogramas B usados como referencias).
- Codificación entrópica CAVLC y CABAC.
- Matrices de cuantificación personalizadas.
- Intra: todos los tipos de macrobloques (16x16, 8x8, 4x4, y PCM con todos los tipos de predicciones).
- Inter P: todas las particiones disponibles (desde 16x16 hasta 4x4).
- Inter B: particiones desde 16x16 hasta 8x8 (incluyendo skip/direct).
- Codificación de material entrelazado (utilizando el método MBAFF).
- Múltiples fotogramas de referencia.
- Modos de control de tasa de bits: cuantificador constante, calidad constante, bitrate medio de una y múltiples pasadas.
- Detección de cambios de escena.
- Multihilo a nivel de fotograma y a nivel de *slice*.
- Modo *lossless* de codificación sin perdidas.
- Optimizaciones psicovisuales para conseguir la mayor calidad de imagen posible: cuantificación adaptativa, psy-RD y psy-trellis.
- Zonas, para codificar diferentes partes del video con diferentes calidades.
- Soporte para los perfiles *Base*, *Main*, *High*, *High 10*, *High 4:2:2* y *High 4:4:4 Predictive*.

x264 es gratuito y su código fuente accesible. Esto unido al soporte para los perfiles *High 10*, *High 4:2:2* y *High 4:4:4 Predictive* lo convierte en el candidato ideal para ser utilizado como codificador H.264 en este proyecto.

Debido a que las variables internas del programa deben cambiar (ser mayores, para poder contener muestras de más de 8 bits) si se quiere codificar utilizando los perfiles *High 10*, *High 4:2:2* o *High 4:4:4 Predictive*, los desarrolladores decidieron que esta opción fuera en tiempo de compilación [10]. Esto implica que distinto número de bits por muestra requiere diferentes ejecutables. En la Figura 4.1 y la Figura 4.2 puede verse el aspecto que presentan la versión compilada con soporte para perfiles de 8 bits y 10 bits, respectivamente. Se ha resaltado la opción de tiempo de compilación en rojo.

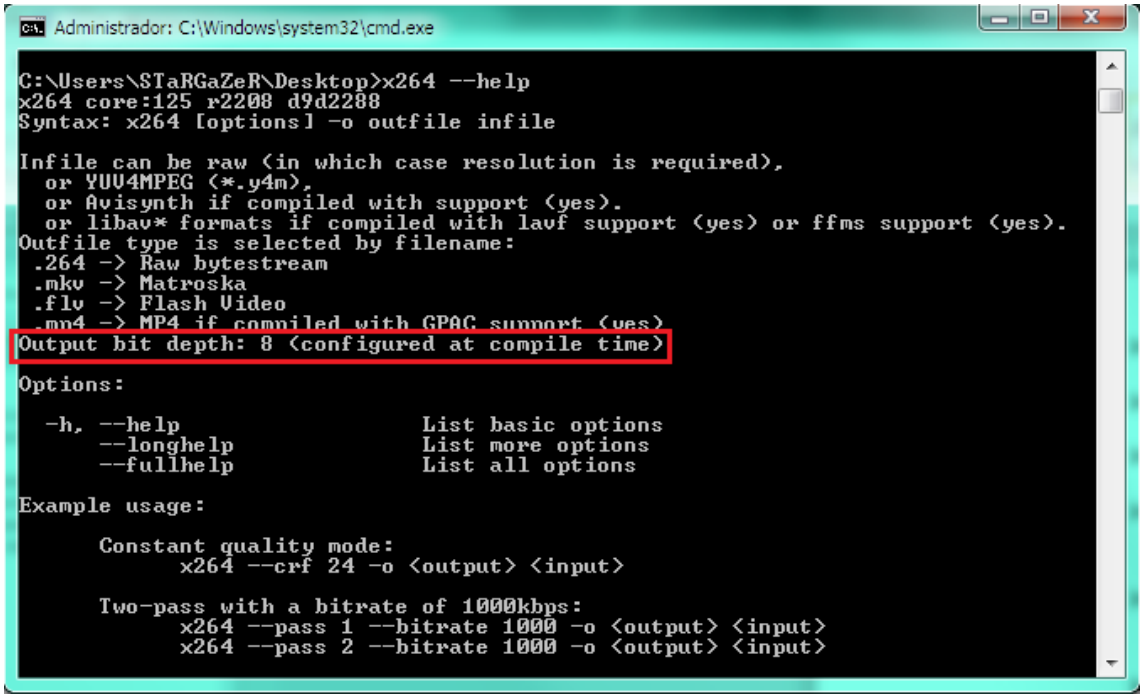

Figura 4.1 - x264 compilado para codificar vídeo de 8 bits por muestra

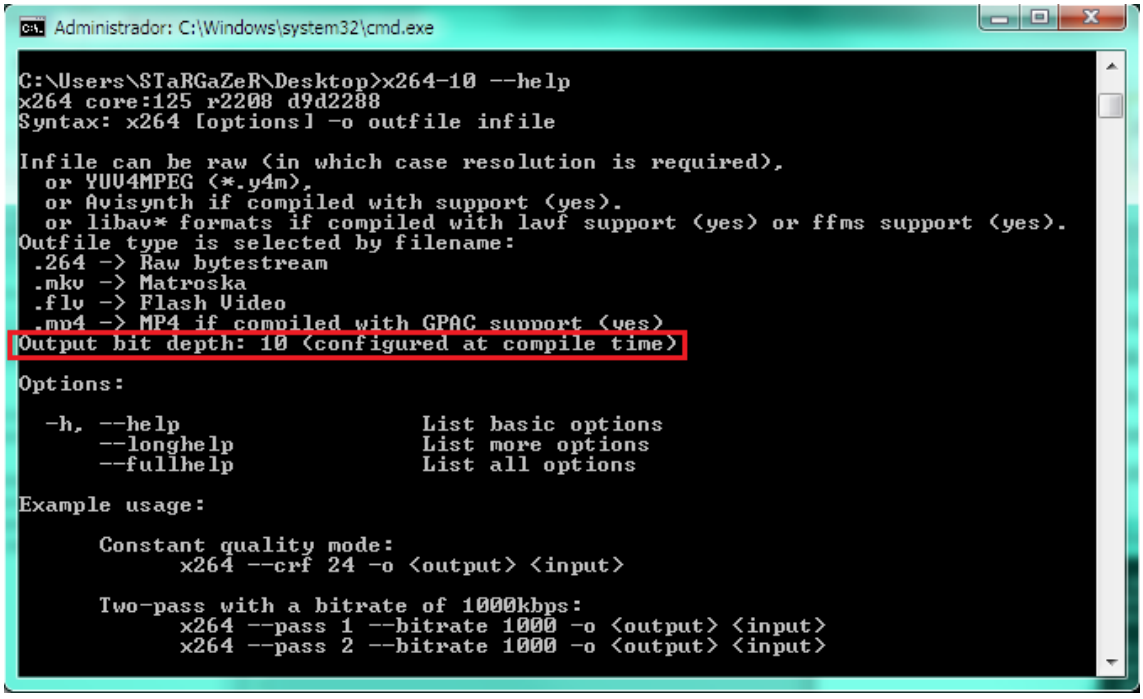

Figura 4.2 - x264 compilado para codificar vídeo de 10 bits por muestra

El proceso de codificación utilizando x264 se resume en los siguientes pasos:

1) Mediante línea de comandos se le pasan al codificador una serie de parámetros, en los que se incluyen el fichero que se quiere codificar, el tipo de fichero de destino y las opciones de codificación con las que se quiere codificar.

2) x264 realiza la codificación, mostrando un indicador de progreso (Figura 4.3).

3) Al finalizar, se muestran una serie de estadísticas, algunas de las cuales solo aparecerán si así lo ha indicado el usuario en la línea de comandos incial (Figura 4.4).

Figura 4.3 - Progreso durante la codificación con x264

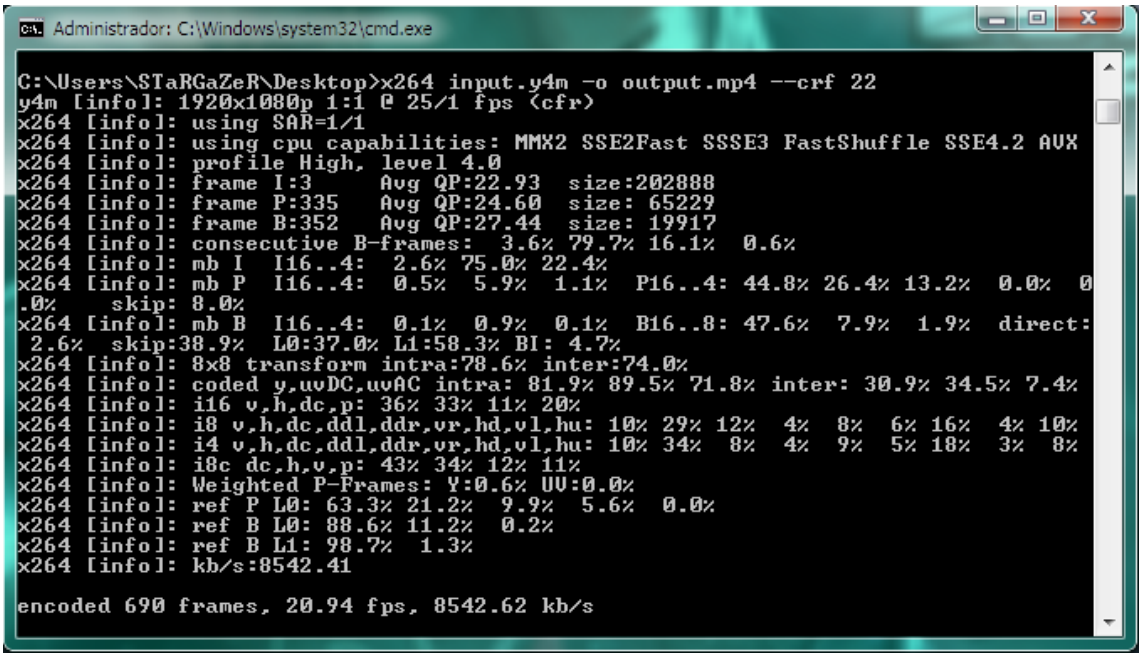

Figura 4.4 - Estadísticas mostradas por x264 al finalizar una codificación

4.2 FFmpeg

FFmpeg es un proyecto de software libre licenciado bajo licencia GNU GPL que contiene una serie de librerías utilizadas para realizar distintas funciones multimedia, como codificación y descodificación de video y audio. Muchos otros proyectos de software libre utilizan dichas librerías, como *FFDShow* [19] y *LAV Filters* [20], este último utilizado en este proyecto fin de carrera.

Al igual que sucede con x264 el desarrollo del proyecto es un proceso constante. Dicho código puede consultarse y descargarse en [21]. El proyecto se divide a su vez en dichas librerías más pequeñas con un propósito concreto, entre las que destacan:

- libavformat: Separar las distintas pistas de audio y video contenidas en los ficheros multimedia. Por ejemplo, si se tiene un fichero *.avi* esta librería se encargaría de abrir el fichero y separar los distintos *bitstreams* de audio y video, para después entregárselos a los decodificadores correspondientes. Es la base para el resto de librerías y soporta un enorme número de formatos diferentes.
- libavcodec: Codifica y descodifica una enorme cantidad de *bitstreams* de audio y vídeo, entre los que se encuentran los más usados como el MPEG-1, MPEG-2, MPEG-4 ASP o H.264. Suele usarse en conjunción con libavformat.
- libavfilter: Contiene multitud de filtros para filtrar tanto video como audio.

Estas librerías pueden compilarse y usarse independientemente unas de otras.

Además de las distintas librerías el proyecto provee de un ejecutable llamado FFmpeg (del cual toma su nombre) en el cual están incluidas todas las librerías anteriormente citadas. Es una aplicación de consola, a la que según se le pasen unos parámetros u otros puede llegar a realizar un gran número de acciones. Un ejemplo de dichas acciones podría ser leer video en formato MPEG-2 de un fichero *.mpg* y codificarlo de nuevo en formato H.264 dentro de un fichero *.mp4*, aplicándole un filtro que aumente un 10% la saturación del color.

FFmpeg es capaz de decodificar ficheros codificados en cualquier perfil H.264 incluyendo *High 10*, *High 4:2:2* y *High 4:4:4 Predictive*, por lo que se le ha elegido como descodificador H.264 en este proyecto. Una vez codificadas las distintas secuencias de prueba en el Capítulo 6, se utiliza FFmpeg para descodificarlas a ficheros *.yuv* o *.y4m* y así poder realizar el estudio de eficiencia de codificación con dichos ficheros y los ficheros *.yuv* o *.y4m* originales.

En la Figura 4.5 puede verse el mensaje de inicio de la aplicación.

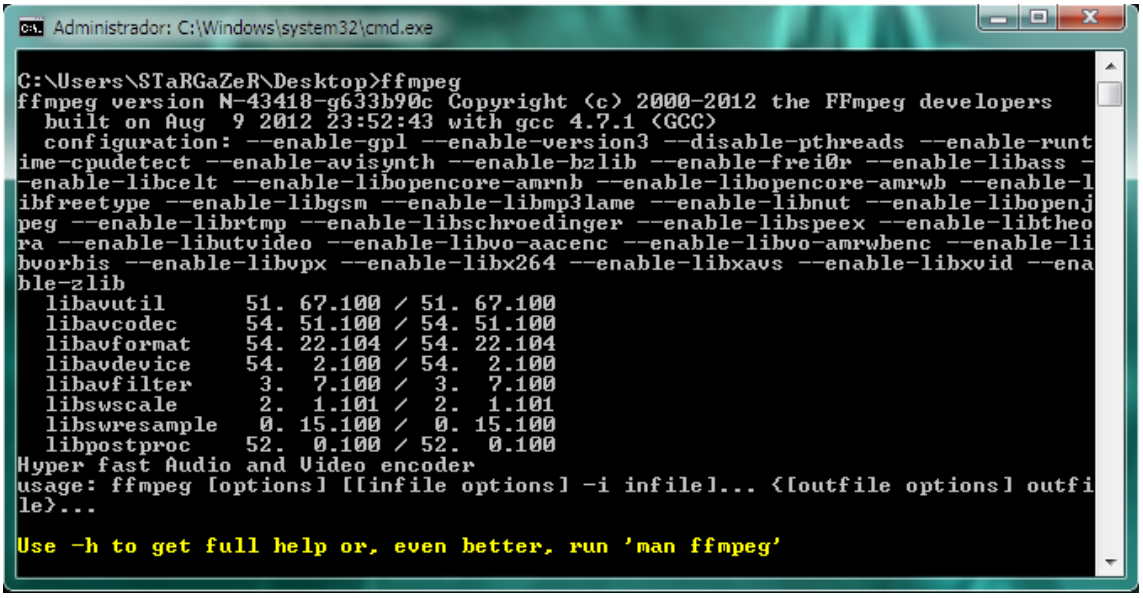

Figura 4.5 - Mensaje de inicio de FFmpeg

FFmpeg posee un gran número de parámetros, la gran mayoría de los cuales no son aplicables a este proyecto. La línea de comandos básica que se utiliza para descodificar los ficheros resultantes de la codificación con x264 es la siguiente:

ffmpeg -i entrada.mp4 salida.yuv

Donde el parámetro **-i** indica el fichero de entrada y **salida.yuv** indica el fichero de salida, donde con la extensión se le indica en qué formato se quiere. En la figura Figura 4.6 puede verse dicho proceso de descodificación con el fichero de prueba *akiyo.mkv*.

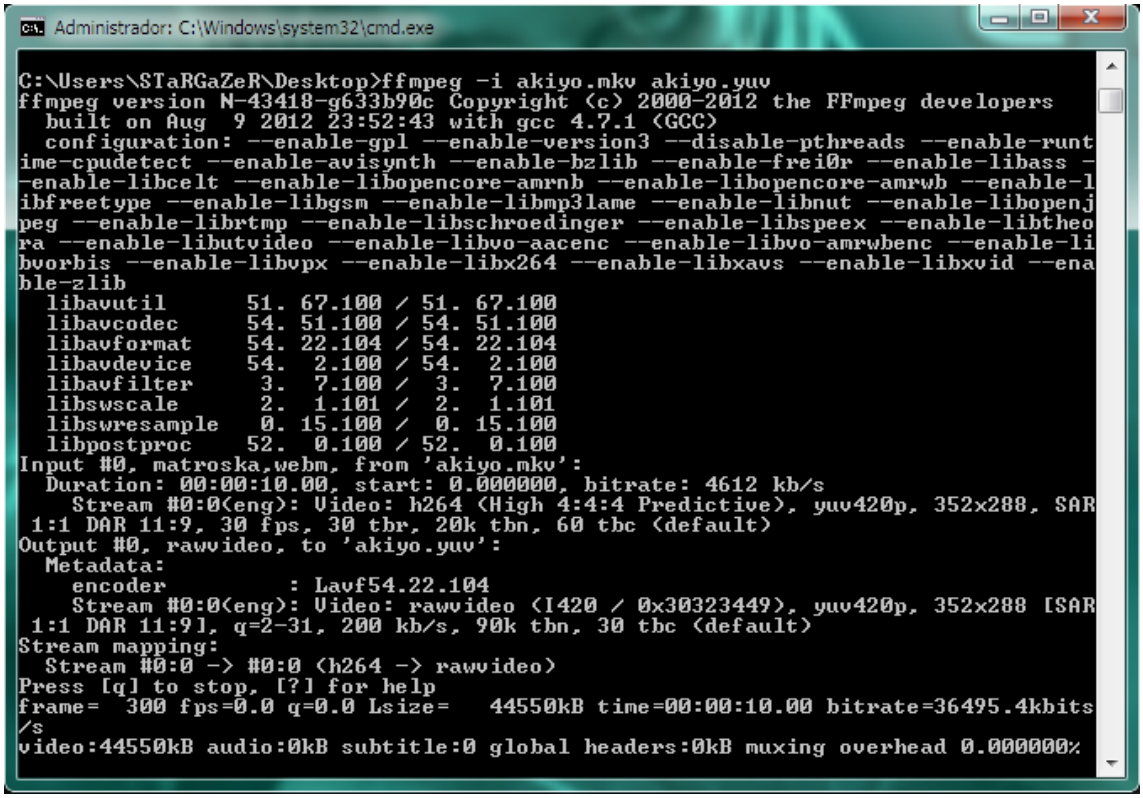

Figura 4.6 - Proceso de descodificación con FFmpeg

4.3 Microsoft Visual Studio 2010

Se ha utilizado la herramienta de desarrollo integrado Visual Studio 2010 de Microsoft tanto para el desarrollo del programa de análisis en el lenguaje de programación C como para la búsqueda de posibles fallos o *bugs*, gracias a su potente módulo de búsqueda de fallos o *debug*. Se ofrece una visión general de la aplicación en la Figura 4.7.

También se ha utilizado para compilar el programa en dos ejecutables distintos:

- Versión de 32 bits para sistemas operativos Windows de 32 bits.
- Versión de 64 bits para sistemas operativos Windows de 64 bits.

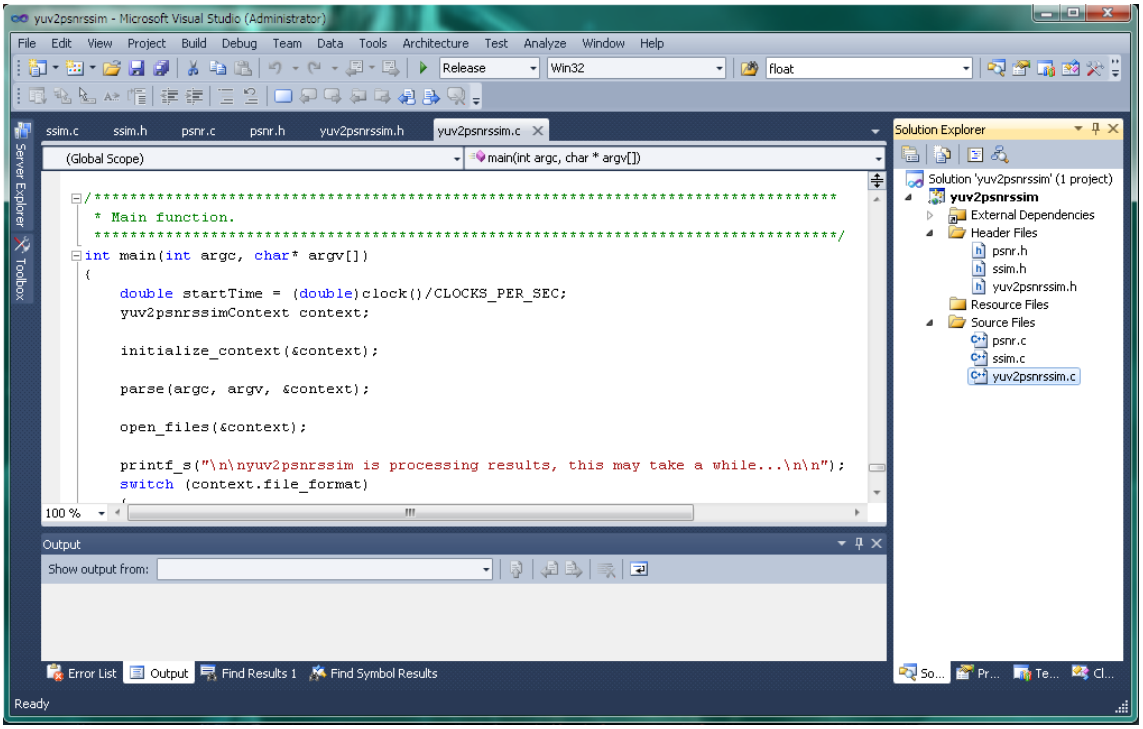

Figura 4.7 - Pantalla general de Microsoft Visual Studio 2010

4.4 Google Code

Durante el desarrollo del programa, el código se ha alojado en un repositorio web en la página para desarrolladores de Google Code [22], usando el sistema de control de versiones Subversion [15].

El objetivo por el cual se ha utilizado este sistema es guardar un historial de todos los cambios realizados a todos los ficheros y directorios del proyecto. Cada cambio al código que se sube al repositorio tiene un número de revisión diferente, y cada revisión se puede descargar y tener identificada de manera independiente de las demás. Esto facilita enormemente el desarrollo de cualquier programa, ya que se pueden revisar versiones antiguas del código si es necesario para solucionar errores no existentes anteriormente. También permite trabajar con el código en varios ordenadores diferentes sin problemas de sincronización, ya que para acceder a los cambios que se hagan en otro lugar y se suban al repositorio no hay más que sincronizar la copia local con la del repositorio para así descargar cualquier modificación que se halla podido hacer.

Google Code es simplemente un *hosting* para proyectos de código abierto que utilizan Subversion y otros sistemas de control de versiones. El código del programa creado para este proyecto se ha alojado en dicho *hosting*. Provee además de una interfaz web

para acceder al listado de versiones del programa (Figura 4.8), y dentro de cada una ver los cambios en el código, de una manera intuitiva gracias al uso de colores (Figura 4.9).

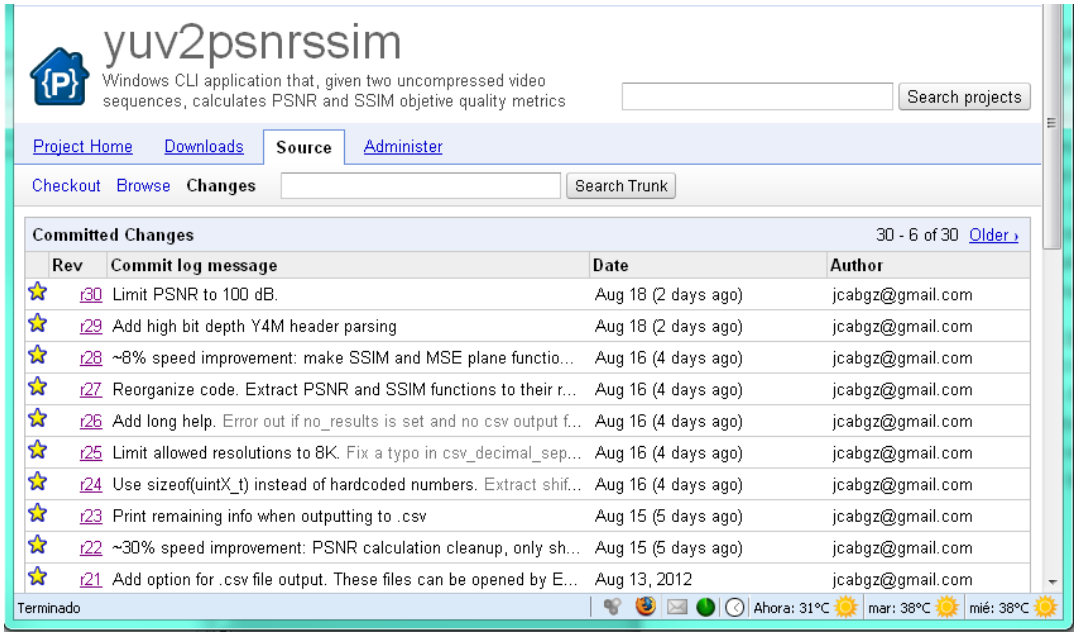

Figura 4.8 - Interfaz web de Google Code, listado de revisiones

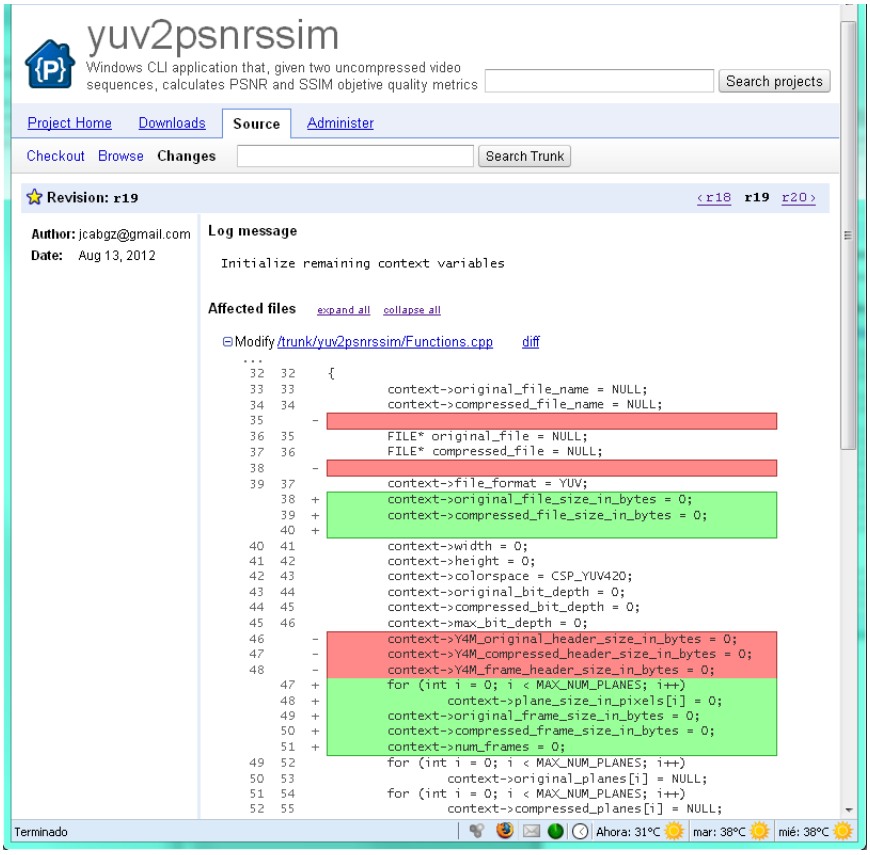

Figura 4.9 - Interfaz web de Google Code, detalle de revisión con colores indicando cambios en el código

4.5 TortoiseSVN

TortoiseSVN [23] es un programa para manejar repositorios Subversion desde Windows, con total integración en Windows Explorer. TortoiseSVN se ha utilizado para manejar el repositorio de código alojado en Google Code, subiendo modificaciones a dicho código y actualizando la copia local con cambios hechos en otro lugar cuando ha sido necesario.

Gracias a la integración con Windows Explorer dichas tareas son sencillas, utilizando los botones “SVN Commit” y “SVN Update” del menú desplegable de la carpeta del programa de análisis (Figura 4.10).

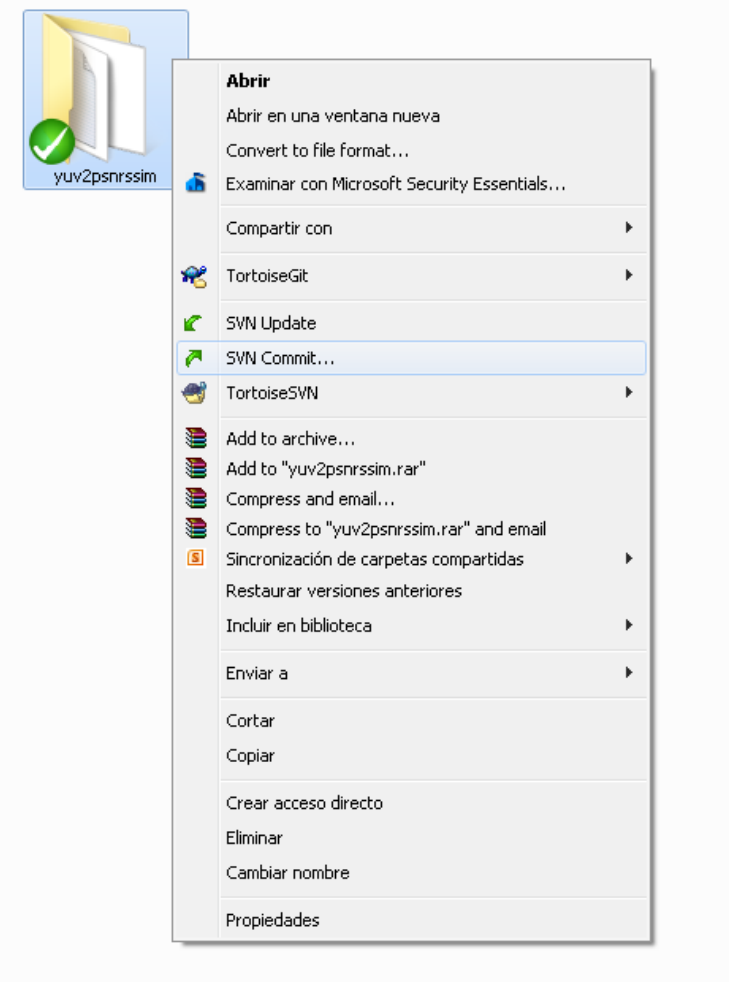

Figura 4.10 - Menú desplegable en Windows Explorer con opciones de gestión SVN gracias a TortoiseSVN

4.6 GraphStudioNext y LAV Filters

Para el estudio del rendimiento de descodificación de los distintos perfiles H.264 se ha utilizado el filtro de descodificación de vídeo incluido en la suite de filtros LAV Filters [20], que utilizan el *framework* de reproducción de Windows DirectShow. Dicho filtro utiliza la librería de descodificación *libavcodec* del proyecto FFmpeg, que como se ha explicado en el apartado 4.2 soporta la descodificación de cualquier perfil del estándar H.264.

Los elementos básicos de una aplicación DirectShow (filtros, conexiones y gráficos de filtros) pueden ser representados gráficamente como cajas con pines de entrada y salida conectados unos con otros formando un gráfico. Existen varias aplicaciones capaces de diseñar y visualizar dichos gráficos, entre la que hay que destacar GraphEdit, diseñada por Microsoft. Para este proyecto se ha utilizado la herramienta GraphStudioNext, ya que aparte de proporcionar todas las funciones que proporciona GraphEdit posee una función de medida de rendimiento ideal para los propósitos de este proyecto. En la Figura 4.11 se puede ver un gráfico DirectShow utilizando LAV Filters para descodificar el fichero *akiyo.mkv* mientras se mide el rendimiento en FPS (*Frames Per Second*, fotogramas por segundo).

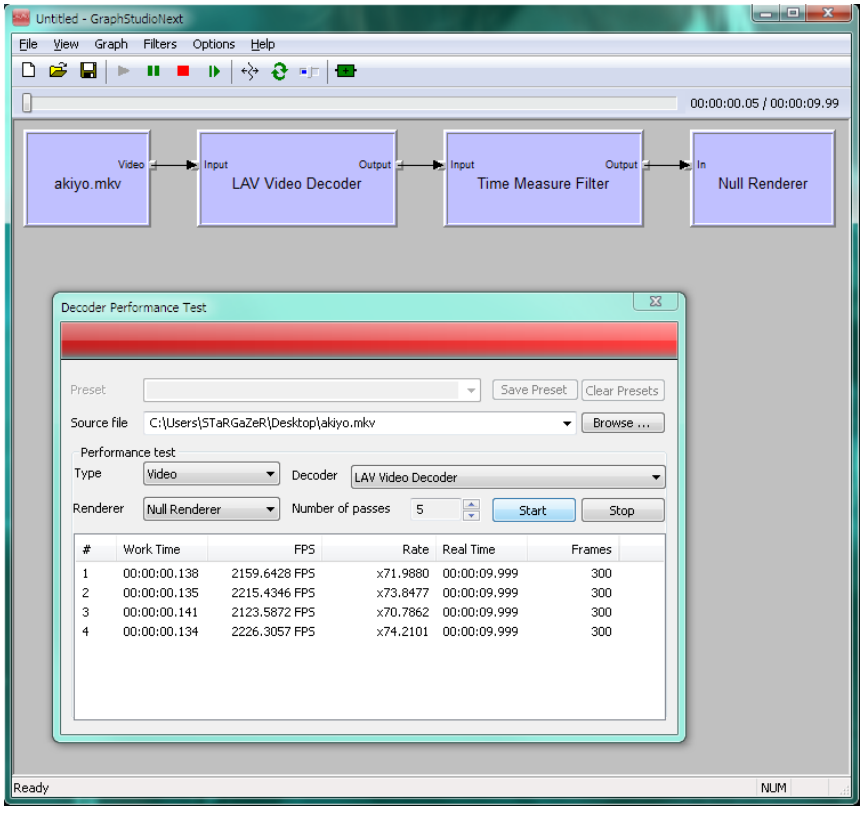

Figura 4.11 - Análisis de rendimiento utilizando GraphStudioNext y LAV Filters

Capítulo 5 - Implementación del programa de análisis

En este capítulo se detalla el programa desarrollado como parte del proyecto para el análisis de calidad objetiva. El nombre elegido es **yuv2psnrssim**. Es una aplicación de consola para sistemas operativos Windows, con versión tanto de 32 bits como de 64 bits. Su objetivo es calcular una serie de medidas objetivas de calidad a partir de dos ficheros de video sin comprimir. Para cada fotograma se calculan las siguientes medidas:

- PSNR Y
- PSNR Cb
- PSNR Cr
- PSNR combinado. Esta medida pretende condensar los PSNR de los tres planos de cada fotograma en una sola medida. No existe una fórmula universal para el cálculo del PSNR combinado de una imagen en color, así que se ha optado por implementar la propuesta contenida en [24]. En ella los planos de diferencia de color tienen un peso de un 10% cada uno, y la luminancia un 80%. Por ello la fórmula utilizada es:

  PSNR combinado = PSNR Y * 0,8 + PSNR Cb * 0,1 + PSNR Cr * 0,1

- SSIM Y

Para la secuencia completa se calculan las siguientes medidas:

- PSNR Y promedio: media del PSNR Y de todos los fotogramas.
- PSNR Cb promedio: media del PSNR Cb de todos los fotogramas.
- PSNR Cr promedio: media del PSNR Cr de todos los fotogramas.
- PSNR combinado promedio: media del PSNR combinado de todos los fotogramas.
- SSIM Y promedio: media del SSIM Y de todos los fotogramas.

Para todas las medidas de PSNR se fija un máximo de 100 dB, ya que cuanto más similares sean las imágenes menor es su MSE, y por tanto el PSNR tiende a infinito.

Dichas medidas son necesarias para realizar el estudio de eficiencia de codificación en el Capítulo 6. Respecto a los ficheros sin comprimir, el programa es capaz de analizar aquellos que tengan las siguientes características:

- Desde 8 hasta 16 bits por muestra.
- Submuestreo de croma 4:2:0, 4:2:2 o 4:4:4.

La guía de los distintos parámetros que acepta el programa se encuentra en el ANEXO I. El código fuente completo puede consultarse y descargarse en [25].

5.1 Ficheros de entrada

5.1.1 YUV

El primer formato de entrada admitido es el formato YUV. Este formato guarda los datos de cada pixel en bruto, serializados. Dependiendo del número de bits de cada muestra el formato del archivo difiere ligeramente:

- 8 bits por muestra: cada muestra de Y, Cb y Cr es representada por un byte (8 bits). El fichero no es más que un flujo binario, por lo que los datos de video vienen serializados siguiendo el siguiente formato:
    - Todas las muestras del plano de luminancia del primer fotograma se colocan primero, en orden de izquierda a derecha y de arriba abajo.
    - Todas las muestras del plano de diferencia de color Cb se colocan a continuación, en orden de izquierda a derecha y de arriba abajo.
    - Por último todas las muestras del plano de diferencia de color Cr se colocan a continuación, en orden de izquierda a derecha y de arriba abajo.

  Una vez serializados los datos del primer fotograma se repite el proceso tantas veces como fotogramas tenga el fichero. Por lo tanto el tamaño total del fichero viene dado por:

Tamaño (bits) = 8 bits * ancho * alto * S * nº fotogramas

Donde **S** depende del submuestreo de color utilizado: 1,5 para 4:2:0, 2 para 4:2:2 y 3 para 4:4:4.

- De 8 hasta 16 bits por muestra: cada muestra de Y, Cb y Cr es representada por **dos** bytes (16 bits), debido a que la unidad de datos más pequeña que manejan los ordenadores es el byte. Esto supone un problema, ya que si el número de bits por muestra es por ejemplo 10, no está claro cómo colocar esos 10 bits dentro de los 16 disponibles. El criterio seguido tanto por x264 como por FFmpeg es colocar los bits de la muestra en los bits menos significativos, dejando a cero el resto de bits. Por ello, si se trabaja con un fichero de 10 bits por muestra los 6 bits más significativos estarán a cero, y a continuación estarán los bits de la muestra. Este criterio es el que sigue también la aplicación de análisis. La serialización es la misma que en el caso de 8 bits por muestra, y el tamaño total del fichero será:

  Tamaño (bits) = 16 bits * ancho * alto * S * nº fotogramas

  Tomando **S** el mismo valor que antes: 1,5 para 4:2:0, 2 para 4:2:2 y 3 para 4:4:4.

Esta diferencia de formato es importante, ya que dependiendo de la cantidad de bits por muestra se debe leer el fichero de una manera o de otra.

Como ya se ha comentado, en este tipo de ficheros los datos están en bruto. No hay ninguna cabecera que indique el tamaño del fotograma, ni los bits por muestra, ni el tipo de submuestreo de crominancia, si lo hay. Por ello al tratar con dichos ficheros es responsabilidad del usuario introducir estos datos correctamente para la correcta lectura de los mismos.

5.1.2 Y4M

El segundo formato de entrada admitido es el formato Y4M [26]. Contiene exactamente los mismos datos que el formato YUV pero con las siguientes diferencias:

- Al principio del archivo se encuentra una cabecera, que informa de las características del vídeo sin comprimir contenido en el archivo. De todos los datos que puede contener la cabecera solo interesan cuatro de ellos para este proyecto: anchura y altura del video, en pixeles, tipo de submuestreo de color y número de bits por muestra. Al final de la cabecera, y para indicar su final, hay siempre un retorno de carro.

- Señalando el comienzo de cada fotograma se coloca la cadena de caracteres “FRAME”, seguida de un retorno de carro. A continuación se colocan los datos del fotograma en el mismo orden que si se tratara de un archivo YUV.

La gran ventaja de usar estos ficheros frente a los YUV es que el usuario no tiene que conocer los datos relativos a resolución, bits por muestra y submuestreo de color, ya que vienen en el propio archivo. Por lo demás, no hay ninguna diferencia. El tamaño total del fichero se calcula de la misma manera que con el formato YUV, sumando el tamaño de las cabeceras, que es variable.

5.2 Funcionamiento interno

A continuación se expone en detalle el funcionamiento interno del programa de análisis. Como debe que procesar ficheros que pueden tener hasta 16 bits por muestra, internamente los fotogramas se almacenan como *arrays* de enteros de 16 bits sin signo. Esto es importante, ya que los ficheros a procesar pueden tener un nº de bits por muestra distinto cada uno, por lo que hay que adaptarlos en este caso (ver punto 4.a). Todos los pasos se ejecutan en orden secuencial:

1) Lectura de la línea de comandos introducida por el usuario y cambio del valor de las variables internas en consecuencia.

2) Apertura de los archivos original y comprimido.

3) Si se trata de archivos en formato Y4M, lectura de la cabecera y cambio de valor de las variables internas en consecuencia.

4) Lectura de los ficheros y procesado. Para el funcionamiento del programa solo es necesario leer un fotograma de cada uno de los archivos a la vez, procesarlo, descartarlo, leer los dos siguientes y así sucesivamente.

    a) Una vez leídos los dos fotogramas, original y comprimido, proceder a su adaptación si sus nº de bits por muestra son diferentes, ya que no se pueden comparar directamente. Esta adaptación consiste en que las muestras de ambos ficheros tengan el mismo margen dinámico, y que solo difieran en su precisión. No tiene sentido comparar directamente muestras de 8 bits, cuyo rango es 0-255, con muestras de 10 bits cuyo rango es 0-1023.

    La adaptación sigue los siguientes pasos, explicados siguiendo el ejemplo anterior donde un fichero tiene precisión de 8 bits por muestra y otro de

10 bits por muestra. Primero se determina cual es el número de bits por muestra más alto de entre los dos ficheros, en este caso 10. El valor de estas muestras no sufre cambios. A continuación las muestras de 8 bits se desplazan la diferencia, en este caso 10 – 8 = 2 bits a la izquierda, o lo que es lo mismo, su valor queda multiplicado por 4. De esta manera el valor de brillo máximo con 10 bits, 940, es el mismo que el de las muestras de 8 bits adaptadas, 235 * 4 = 940, y el valor de brillo mínimo también coincide, 64 para 10 bits y 16 * 4 = 64 para las muestras de 8 bits adaptadas. La única diferencia está en la precisión, ya que las muestras de 10 bits pueden tomar todos los valores entre 64 y 940, mientras que las de 8 bits adaptadas solo pueden tomar valores múltiplos de 4 al tener sus 2 bits menos significativos a cero en todo momento.

b) Una vez adaptados los valores se realizan los cálculos de medidas de calidad PSNR y SSIM como se ha descrito en el Capítulo 3.

c) Se repite el mismo proceso para todos los fotogramas de la secuencia.

5) Presentación de resultados. Se expone en detalle más adelante.

Si en algún punto del programa se produce algún fallo, se muestra un mensaje de error detallando dónde ha fallado y la ejecución termina en ese instante.

5.3 Presentación de resultados

Una vez realizados los distintos cálculos el programa ofrece dos maneras de mostrarlos al usuario:

- Por pantalla a través de la consola. Este modo es útil para ver rápidamente los resultados, pero no muy adecuado para su procesado posterior. Activado por defecto, este modo puede desactivarse mediante el parámetro **-no_results**. En la Figura 5.1 y la Figura 5.2 se muestra un ejemplo del principio y del final de los resultados mostrados en la consola, respectivamente.

- Exportación a formato CSV [27]. Estos archivos están compuestos por texto plano y son importables por el programa Excel. El formato de dichos ficheros es fácilmente conseguible mediante funciones del lenguaje de programación C de escritura en fichero, como *fprintf*. Excel trabaja con estos ficheros sin problemas, lo que facilita su uso posterior para crear gráficos u operar con los datos. Para la realización de las pruebas del proyecto se usa este modo.

Esta opción está desactivada por defecto, activándose mediante **-csv_output_file <nombre_del_fichero>.csv**. Debido a que este formato utiliza en principio comas para separar los distintos valores, se crea un problema al trabajar con cifras decimales en formato europeo, que utiliza la coma como separador decimal. Debido a esto Excel diferencia entre dos tipos de ficheros CSV, aquellos que utilizan la coma como separador decimal (y punto y coma como separador de valores) y los que usan el punto (y coma como separador de valores). Por ello el programa ofrece una opción para cambiar el formato del fichero CSV según se vaya a visualizar en Excel con la configuración de separador decimal europea o americana:

- **-csv_decimal_separator comma** ó **-csv_decimal_separator ,**
- **-csv_decimal_separator dot** ó **-csv_decimal_separator .**

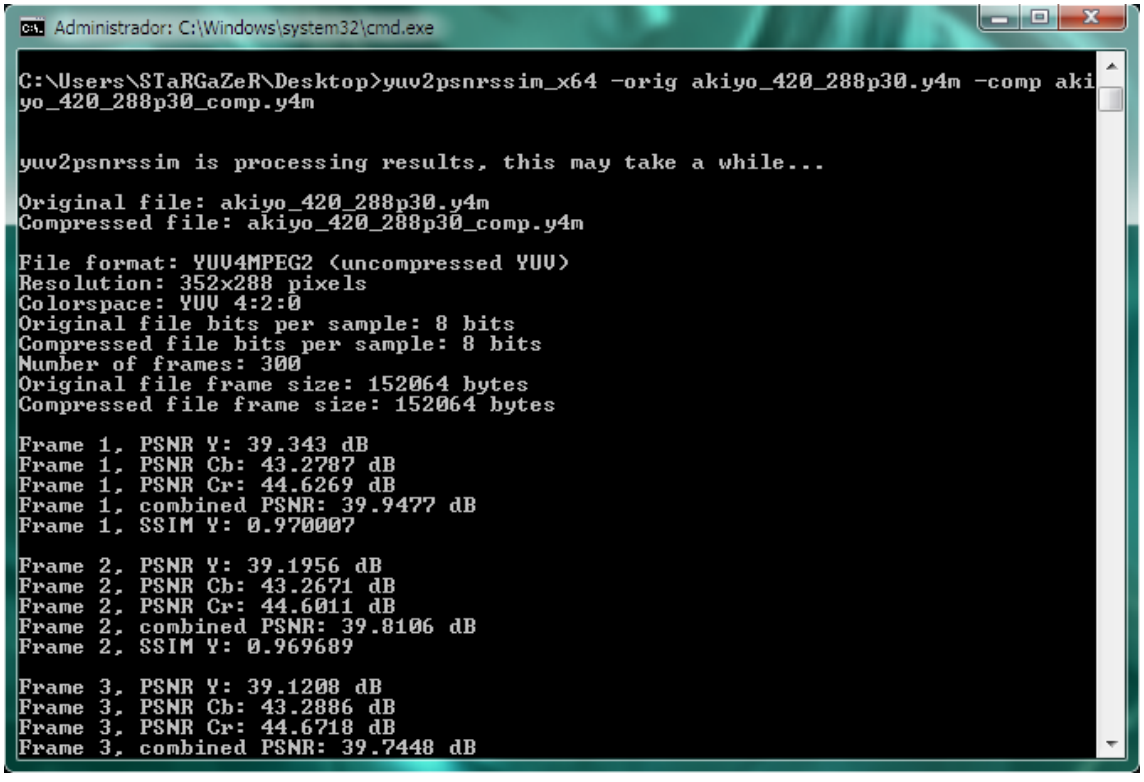

Figura 5.1 - Inicio de la presentación de resultados del programa de análisis

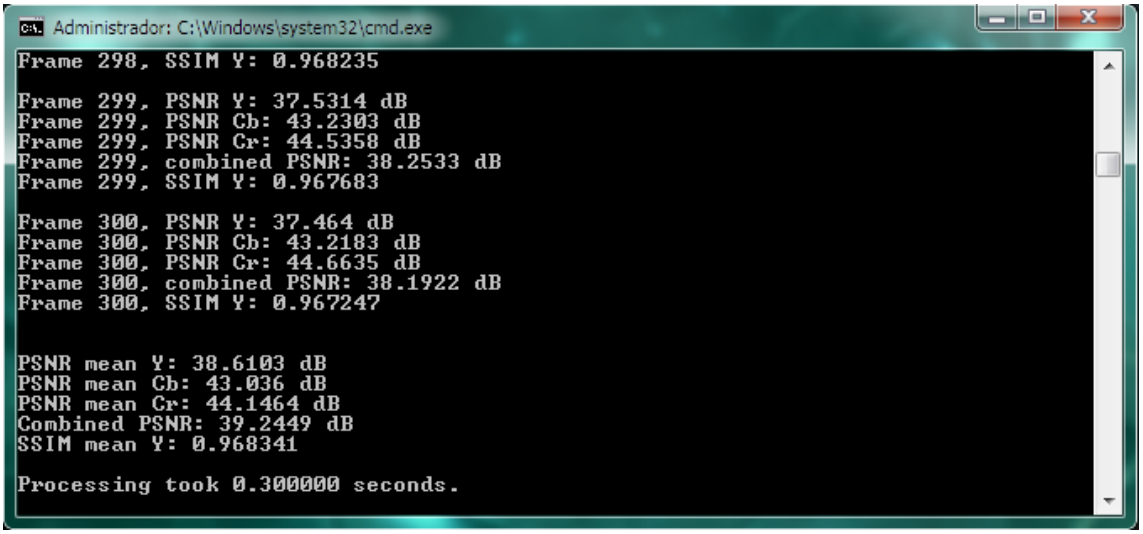

Figura 5.2 - Final de la presentación de resultados del programa de análisis

Capítulo 6 - Pruebas de eficiencia de codificación: 8 vs. 10 bits

En este capítulo se realizan las distintas pruebas de eficiencia de codificación utilizando los perfiles H.264 de 8 y 10 bits de precisión. Para ello se analizan 7 secuencias de video, en formato *.y4m*:

- *Blue_Sky*
- *Sunflower*
- *Rush_Hour*
- *Tractor*
- *Sintel_Trailer*
- *Dinner*
- *Ducks_Take_Off*

Se ha intentado elegir las secuencias de tal manera que cubran un amplio abanico de tipos de video, desde imagen real a animación, con variedad de cantidad de movimiento y detalle, etc. Una descripción más detallada de cada secuencia puede encontrarse en cada apartado individual junto a los resultados. Todas han sido sacadas de página web de la fundación Xiph [28].

El número de bits por muestra de todas las secuencias es 8, y su resolución de 1920x1080 pixeles, con submuestreo de los planos de diferencia de color 4:2:0, excepto la secuencia *Ducks_Take_Off*. En este caso se han probado dos resoluciones, 1280x720 y 3840x2160, y para la resolución de 1280x720 los tipos de submuestreo de planos de diferencia de color 4:2:0, 4:2:2 y 4:4:4.

6.1 Configuración de las pruebas

Para estas pruebas se ha decidido codificar con parámetros tales que produzcan la máxima calidad de imagen para cada una de las tasas de bits analizadas, sin importar la velocidad de codificación obtenida.Para ello se ha utilizado el *preset* x264 más lento, *placebo* (ver

Parámetros *tune y preset* en el ANEXO II). Esto se ha hecho así para mostrar todo el potencial de la codificación con 10 bits de precisión, dejando que el codificador utilice todas las herramientas de las que dispone, aunque ello implique unas velocidades de codificación que en la mayoría de los casos no superan el fotograma por segundo. Previamente se han analizado otros *presets* más veloces, obteniendo resultados similares en todos ellos.

Para la realización de las pruebas se han dado los siguientes pasos:

1) Codificación de la secuencia con x264 a un archivo en formato *.mp4*. Los parámetros utilizados se detallan más adelante. Cabe resaltar que para conseguir los ficheros codificados de 8 y 10 bits lo único que cambia es el ejecutable de x264, como se explicó en el capítulo 4.1. Todos los parámetros son idénticos en ambos casos.

2) Decodificación del resultado de nuevo a un archivo en formato *.y4m* para su posterior análisis, usando FFmpeg con la línea de comandos:

   ffmpeg -i <secuencia>.mp4 <secuencia>.y4m

3) Análisis de los dos ficheros *.y4m*, original y descomprimido, mediante el programa implementado por el alumno, **yuv2psnrssim**. La línea de comandos utilizada es la siguiente:

   yuv2psnrssim -o <secuencia>.y4m -c <secuencia_codificada>.y4m
   -csv_output_file <secuencia>.csv -csv_decimal_separator comma

   Dichos parámetros generan un fichero CSV con los resultados del análisis.

Con los datos almacenados en los ficheros CSV para cada secuencia se ofrecen tres resultados:

- Un gráfico con dos curvas RD (*Rate Distortion*), una codificando con 8 bits de precisión y otra con 10 bits de precisión, con la distorsión medida como PSNR combinado promedio (ver Capítulo 5) de toda la secuencia y la tasa de bits medida en megabits por segundo (Mbps). Para confeccionar las curvas de cada secuencia se han realizado un total de 10 codificaciones extendidas sobre un rango de tasa de bits que varia según la secuencia, de tal manera que proporciona una idea clara de las tendencias de dichas curvas. Como se pretende medir el rendimiento a distintas tasas binarias medias se utiliza el modo de codificación a dos pasadas, que es el único que garantiza que al final de las codificaciones se obtengan las tasas binarias medias especificadas (ver Parámetros relacionados con el control de la tasa de bits en el ANEXO II). Por tanto las líneas de comandos pasadas al codificador x264 para las distintas pruebas son:

x264 <secuencia>.y4m -o <secuencia>.mp4 --preset placebo --tune psnr --bitrate ** --pass 1**

x264 <secuencia>.y4m -o <secuencia>.mp4 --preset placebo --tune psnr --bitrate ** --pass 2**

En caso de estar codificando video 4:2:2 o 4:4:4 se añaden los parámetros **--input-csp i422 --output-csp i422** ó **--input-csp i444 --output-csp i444**. Esto le indica a x264 que use los perfiles *High 4:2:2* y *High 4:4:4 Predictive*, respectivamente. Se utiliza **--tune psnr** para que x264 codifique intentando conseguir siempre el máximo PSNR para la tasa de bits dada.

- Otro gráfico con dos curvas RD, una codificando con 8 bits de precisión y otra con 10 bits de precisión, con la distorsión medida como SSIM Y promedio de toda la secuencia y la tasa de bits medida en megabits por segundo (Mbps). Al igual que en el gráfico anterior para confeccionar la curva de cada secuencia se han realizado un total de 10 codificaciones extendidas sobre un rango de tasa de bits que varia con la secuencia, de tal manera que proporciona una idea clara de las tendencias de dichas curvas.

  Los parámetros pasados a x264 son idénticos al caso anterior con una única diferencia: **--tune ssim**, para que x264 codifique intentando conseguir siempre el máximo SSIM para la tasa de bits dada.

- Un tercer gráfico con la evolución del PSNR combinado a lo largo del tiempo que dure cada secuencia para una tasa de bits determinada. Se ha elegido que esta tasa de bits sea la intermedia de todas las analizadas en cada secuencia. Se muestran 4 curvas: la evolución del PSNR combinado de 8 bits, la evolución del PSNR combinado de 10 bits y una media móvil de 10 puntos por cada una de ellas. Cada punto de estas medias se calcula como la media de los 10 puntos anteriores de la curva, y tienen como objetivo ofrecer una visión más uniforme de los resultados, ya que el PSNR puede variar en gran medida de fotograma en fotograma.

A continuación se presentan las pruebas efectuadas para cada una de las secuencias mencionadas.

6.2 Secuencia *Blue_Sky*

Figura 6.1 - Secuencia *Blue_Sky*

La secuencia *Blue_Sky* consta de 217 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se muestran las copas de dos árboles contra cielo azul. La cámara va rotando en contra de las agujas del reloj. Existe un gran contraste entre los árboles y el cielo, gradiente de azul en el cielo y mucho detalle en las copas de los árboles. Contiene una pequeña cantidad de ruido. En la Figura 6.1 pueden verse capturas de los fotogramas número 1, 108 y 217.

En la Figura 6.2 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.3 se muestra el segundo gráfico, SSIM Y.

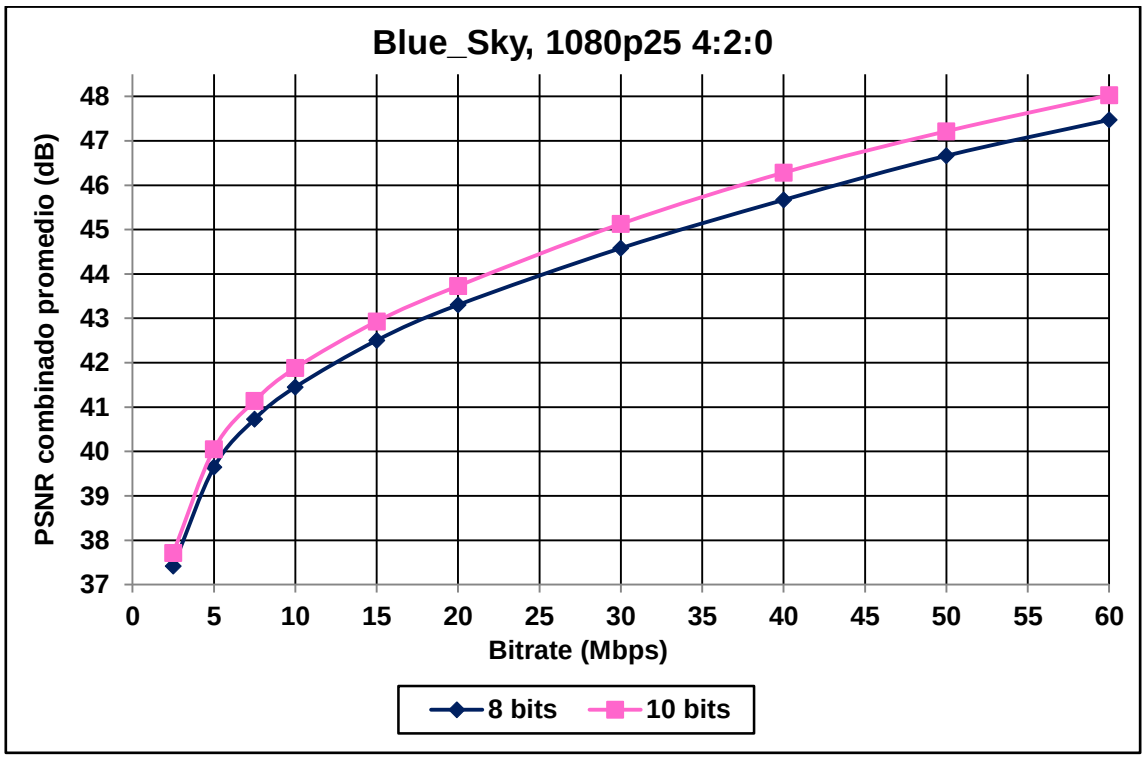

Figura 6.2 - PSNR combinado promedio de la secuencia *Blue_Sky* a diferentes regímenes binarios

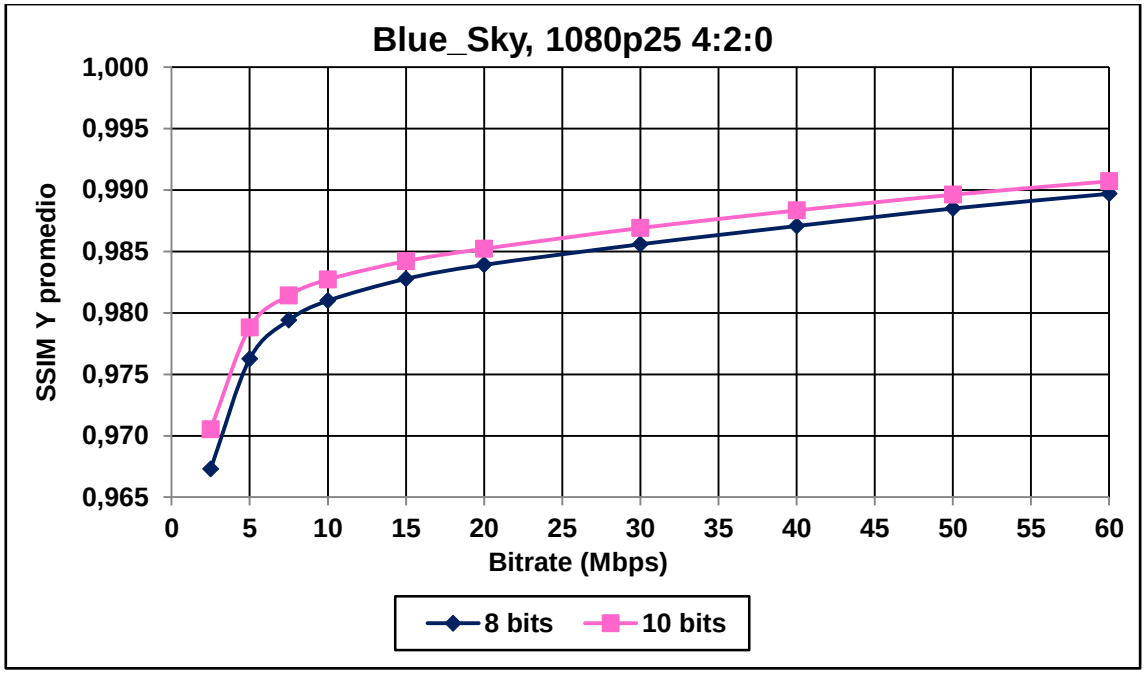

Figura 6.3 - SSIM Y promedio de la secuencia *Blue_Sky* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 30 Mbps, en la Figura 6.4.

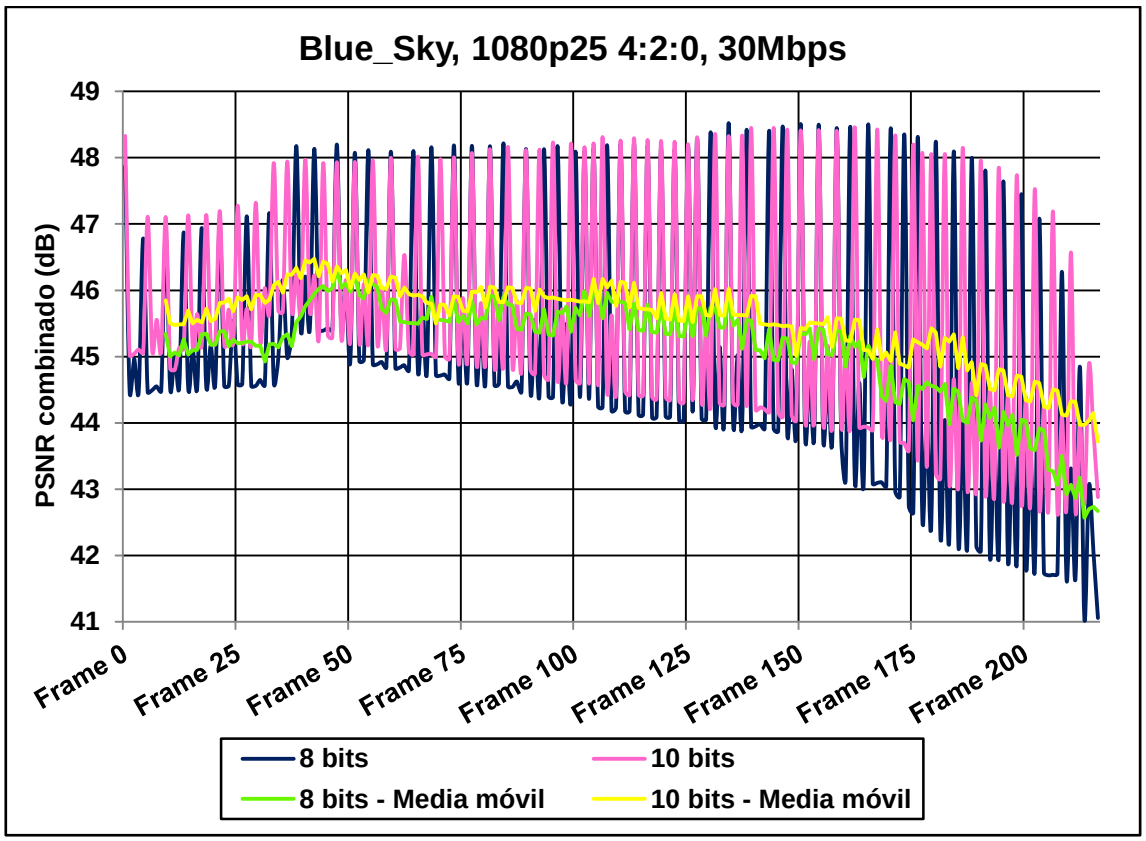

Figura 6.4 - PSNR combinado de la secuencia *Blue_Sky*, evolución temporal

La Figura 6.2 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de entre un 10% y un 15% de tasa de bits para los mismos valores de PSNR, o un aumento de entre 0,3 y 0,6 dB de PSNR para las mismas tasas de bits.

La Figura 6.3 muestra una reducción de entre un 15% y un 25% de tasa de bits para los mismos valores de SSIM.

Analizando la evolución temporal del PSNR en la Figura 6.4 las mayores ganancias se concentran al principio, donde hay una gran cantidad de cielo, con pocas y lentas variaciones en los pixeles, y al final, donde los árboles apenas se mueven.

La gráfica presenta esa forma de diente de sierra debido a que x264 destina más bits a codificar fotogramas tipo I que tipo P, y más a los tipo P que a los tipo B. Utilizando el *preset placebo*, **--keyint** toma el valor por defecto de 250. Como no hay cambios de escena durante la secuencia y ésta dura menos de 250 fotogramas, x264 coloca un único fotograma IDR, el primero de la secuencia. El resto son tipo P o tipo B. Al destinar x264 más bits a los fotogramas tipo P que a los B se genera esa forma de diente de sierra, con los tipo P en los picos y los B en los valles. Este comportamiento se puede variar mediante los parámetros **--ipratio** y **--pbratio**, pero como se ha explicado al principio del capítulo se ha dejado que x264 tome todas las decisiones que considere oportunas para maximizar el PSNR. Dichas decisiones afectan por igual a las codificaciones con 8 y 10 bits de precisión, como demuestra el gráfico, y se repiten a lo largo de todas las secuencias. Se recomienda consultar el ANEXO II para una descripción más detallada de los parámetros más importantes de x264 y sus valores por defecto.

6.3 Secuencia *Sunflower*

Figura 6.5 - Secuencia *Sunflower*

La secuencia *Sunflower* consta de 500 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se muestra una abeja alimentándose en un girasol. La cámara está estática, aunque el girasol se mueve ligeramente. Hay mucho detalle y pocas diferencias de color, con gran cantidad de verde y amarillo. Apenas tiene ruido. En la Figura 6.5 pueden verse capturas de los fotogramas número 1, 250 y 500.

En la Figura 6.6 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.7 se muestra el segundo gráfico, SSIM Y.

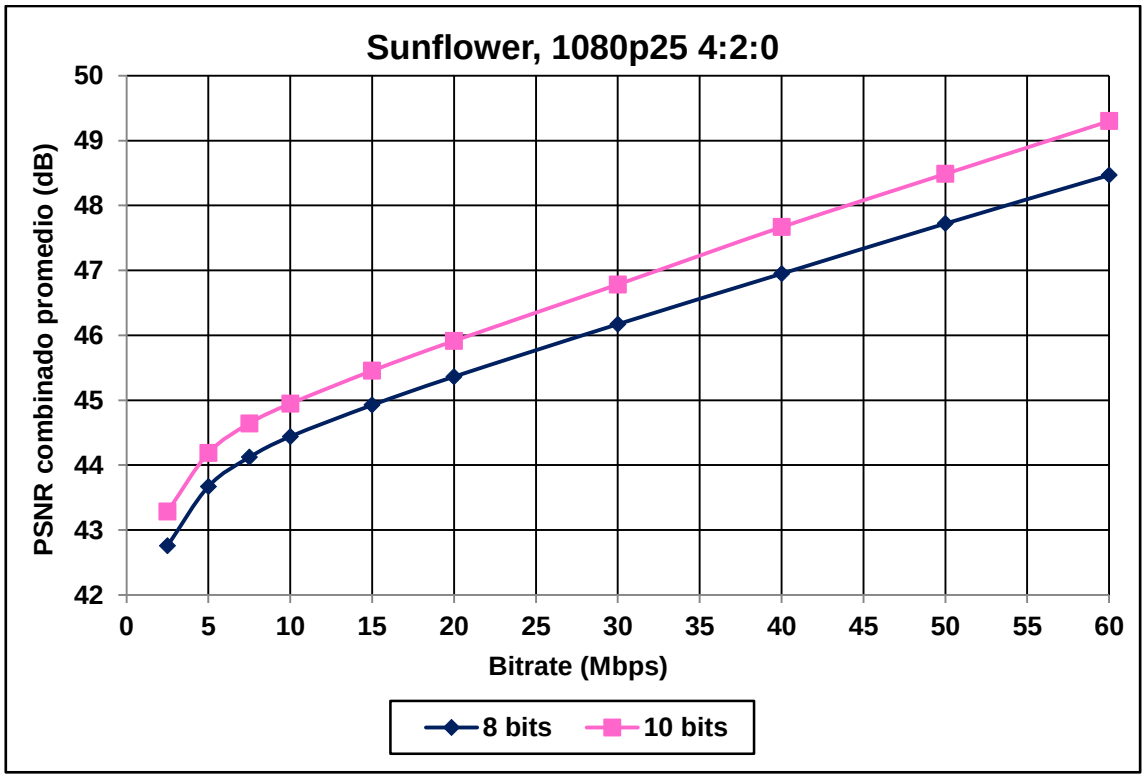

Figura 6.6 - PSNR combinado promedio de la secuencia *Sunflower* a diferentes regímenes binarios

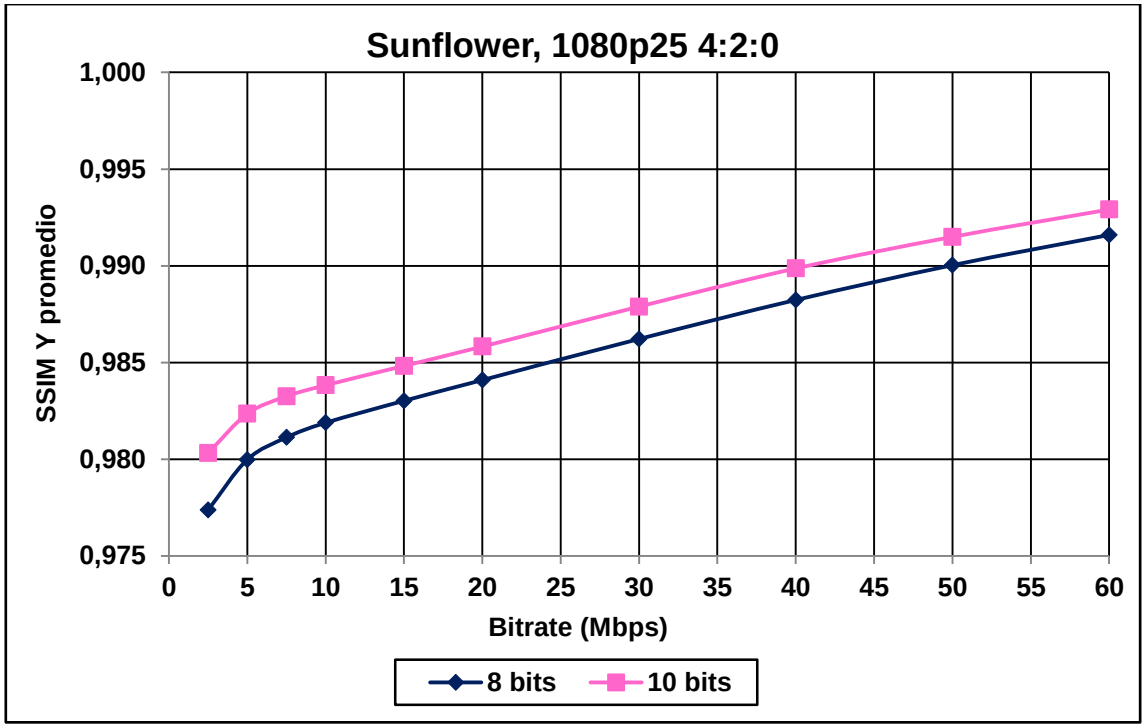

Figura 6.7 - SSIM Y promedio de la secuencia *Sunflower* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 30 Mbps, en la Figura 6.8.

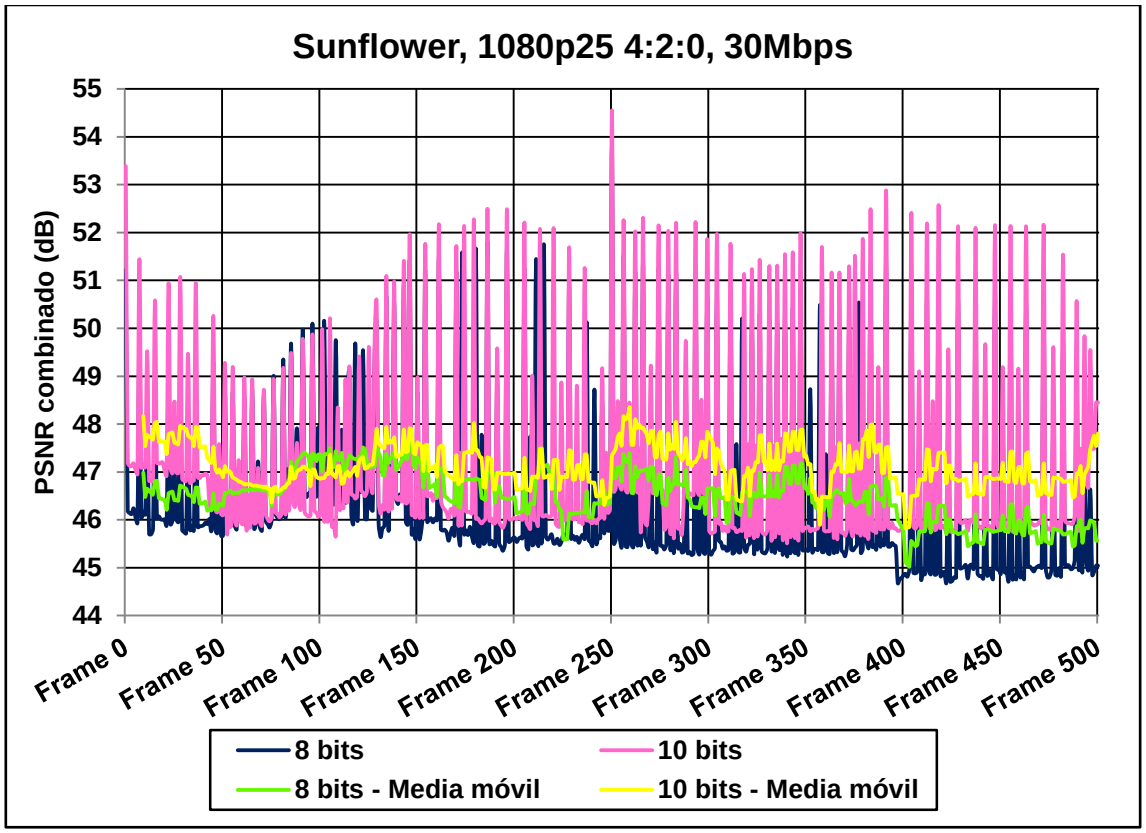

Figura 6.8 - PSNR combinado de la secuencia *Sunflower*, evolución temporal

La Figura 6.6 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de entre un 18% y un 25% de tasa de bits para los mismos valores de PSNR, o un aumento de entre 0,4 y 0,9 dB de PSNR para las mismas tasas de bits.

La Figura 6.7 muestra una reducción de entre un 20% (a tasas altas) y un 50% (a tasas bajas) de tasa de bits para los mismos valores de SSIM.

Analizando la evolución temporal del PSNR en la Figura 6.8 la ganancia está más o menos distribuida a lo largo de toda la secuencia.

La gráfica presenta esa forma de diente de sierra por la misma razón que la secuencia *Blue_Sky*. En este caso x264 ha colocado dos fotogramas IDR, al principio y en el fotograma 250 (picos máximos). El resto de picos corresponden a fotogramas tipo P y los valles a tipo B.

6.4 Secuencia *Rush_Hour*

Figura 6.9 - Secuencia *Rush_Hour*

La secuencia *Rush_Hour* consta de 500 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se muestra la ciudad de Munich en hora punta. La cámara está estática, y hay bastante movimiento de los coches, aunque lento. La toma tiene gran profundidad de campo. Aunque apenas tiene ruido el efecto del calor de los coches y el asfalto distorsiona la imagen. En la Figura 6.9 pueden verse capturas de los fotogramas número 1, 250 y 500.

En la Figura 6.10 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.11 se muestra el segundo gráfico, SSIM Y.

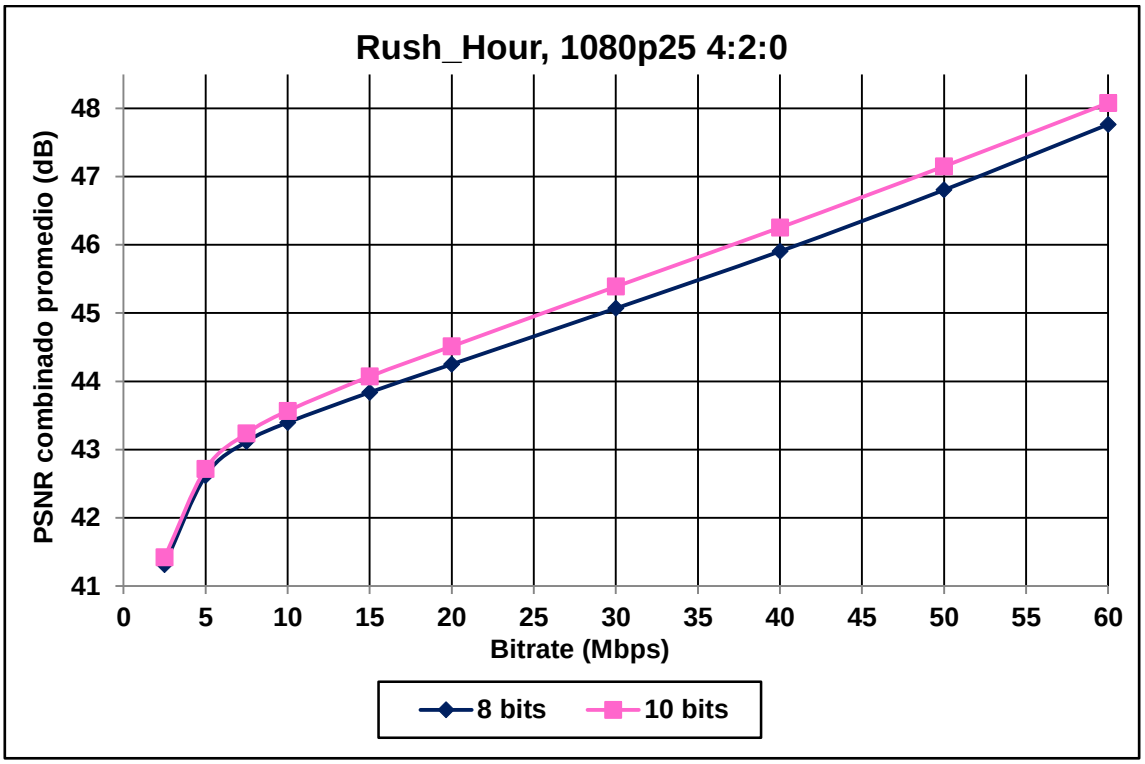

Figura 6.10 - PSNR combinado promedio de la secuencia *Rush_Hour* a diferentes regímenes binarios

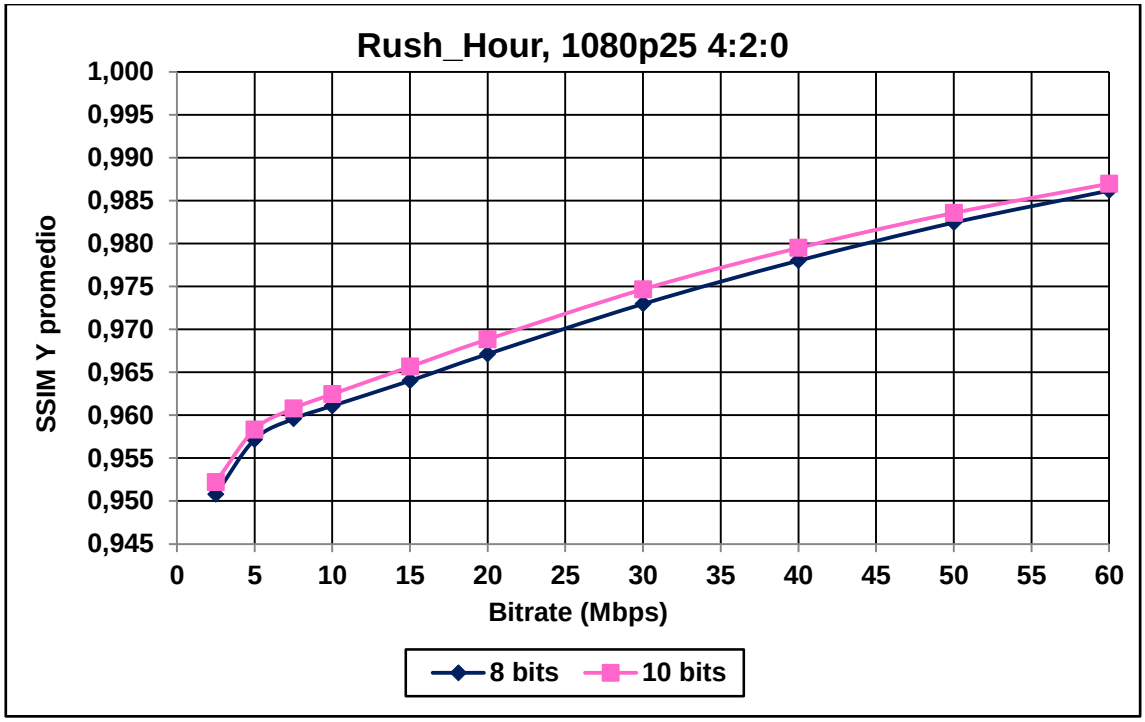

Figura 6.11 - SSIM Y promedio de la secuencia *Rush_Hour* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 30 Mbps, en la Figura 6.12.

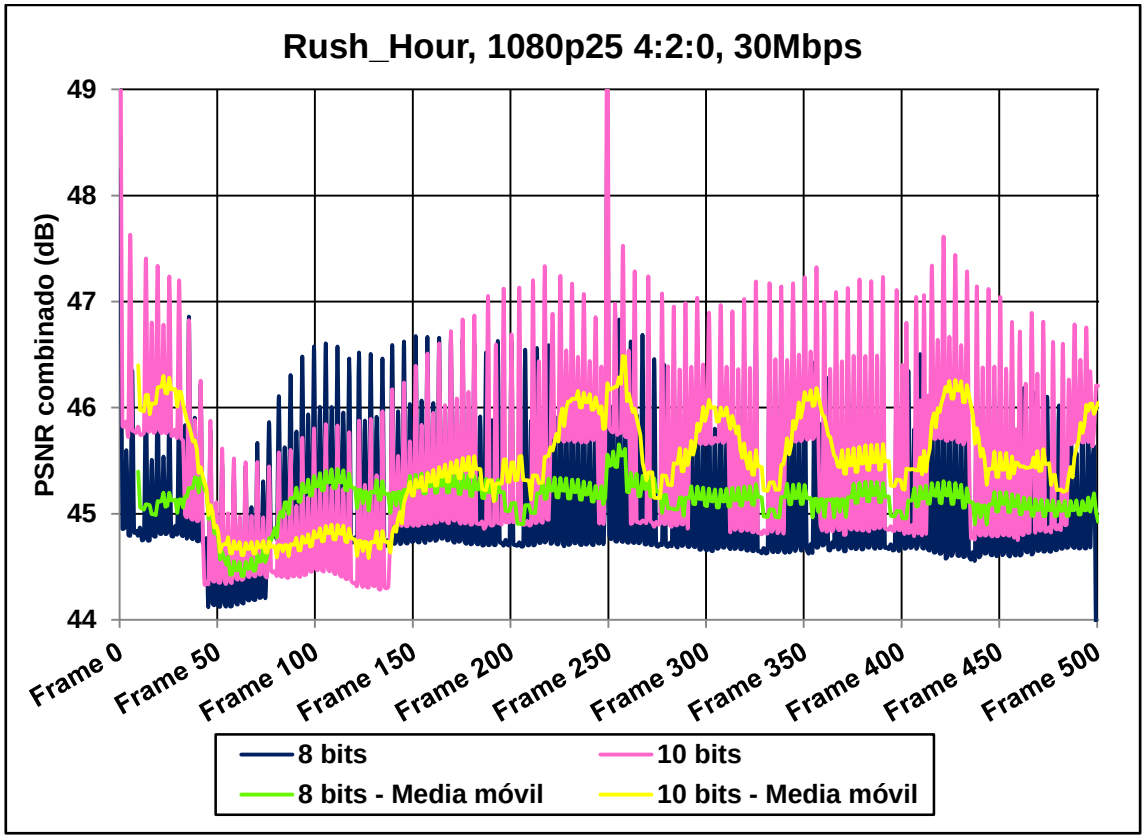

Figura 6.12 - PSNR combinado de la secuencia *Rush_Hour*, evolución temporal

La Figura 6.10 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de aproximadamente un 10% de tasa de bits para los mismos valores de PSNR a tasas altas, no habiendo apenas diferencia para tasas bajas. Esto se traduce en un aumento de 0,3-0,4 dB de PSNR para esas tasas altas de bits.

La Figura 6.11 muestra una reducción de entre un 5% (a tasas altas) y un 10% (a tasas bajas) de tasa de bits para los mismos valores de SSIM.

Analizando la evolución temporal del PSNR en la Figura 6.12 las mayores ganancias se concentran desde aproximadamente la mitad de la secuencia hasta el final, donde la cantidad de movimiento global disminuye ligeramente.

La gráfica presenta esa forma de diente de sierra por la misma razón que las secuencias anteriores. En este caso x264 ha colocado dos fotogramas IDR, al principio y en el fotograma 250 (picos máximos). El resto de picos corresponden a fotogramas tipo P y los valles a tipo B.

6.5 Secuencia *Tractor*

Figura 6.13 - Secuencia *Tractor*

La secuencia *Tractor* consta de 690 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se muestra un tractor trabajando en el campo. La cámara empieza enfocando al tractor, después se centra en la parte de atrás mientras el tractor se mueve y por ultimo se aleja para mostrar una panorámica del terreno. La secuencia contiene gran cantidad de movimiento mientras el tractor está trabajando y la cámara está cerca, y poco movimiento al alejarse. Presenta ruido moderado, bastante visible en el cielo. En la Figura 6.13 pueden verse capturas de los fotogramas número 1, 345 y 690.

En la Figura 6.14 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.15 se muestra el segundo gráfico, SSIM Y.

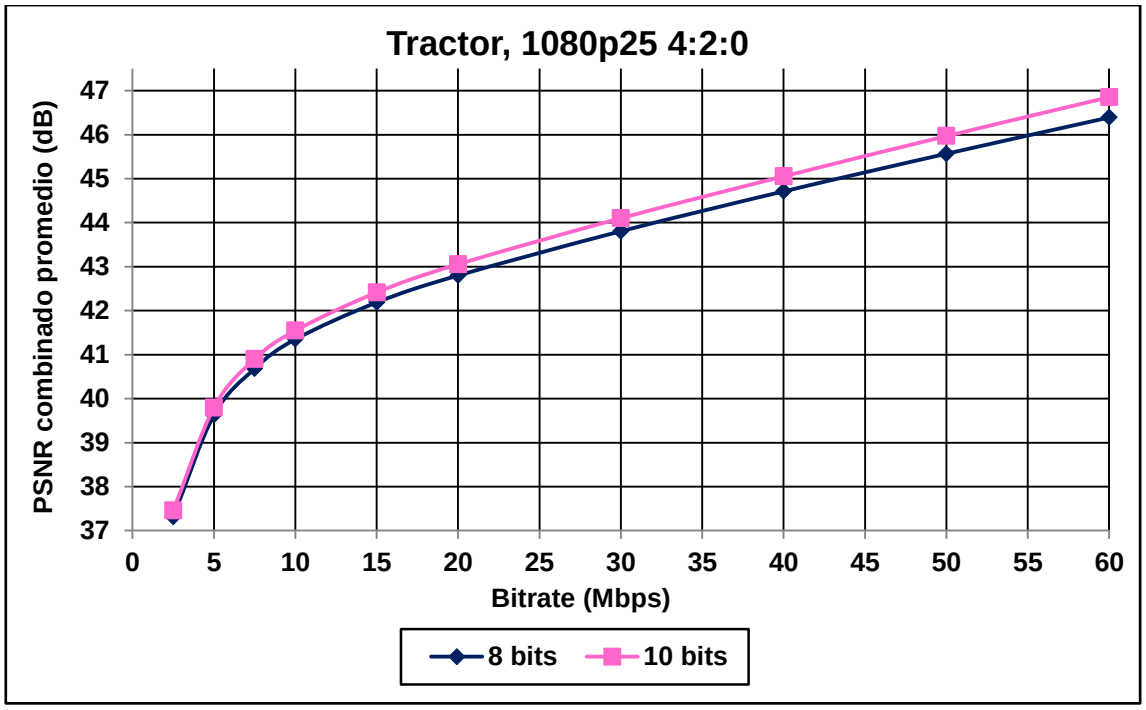

Figura 6.14 - PSNR combinado promedio de la secuencia *Tractor* a diferentes regímenes binarios

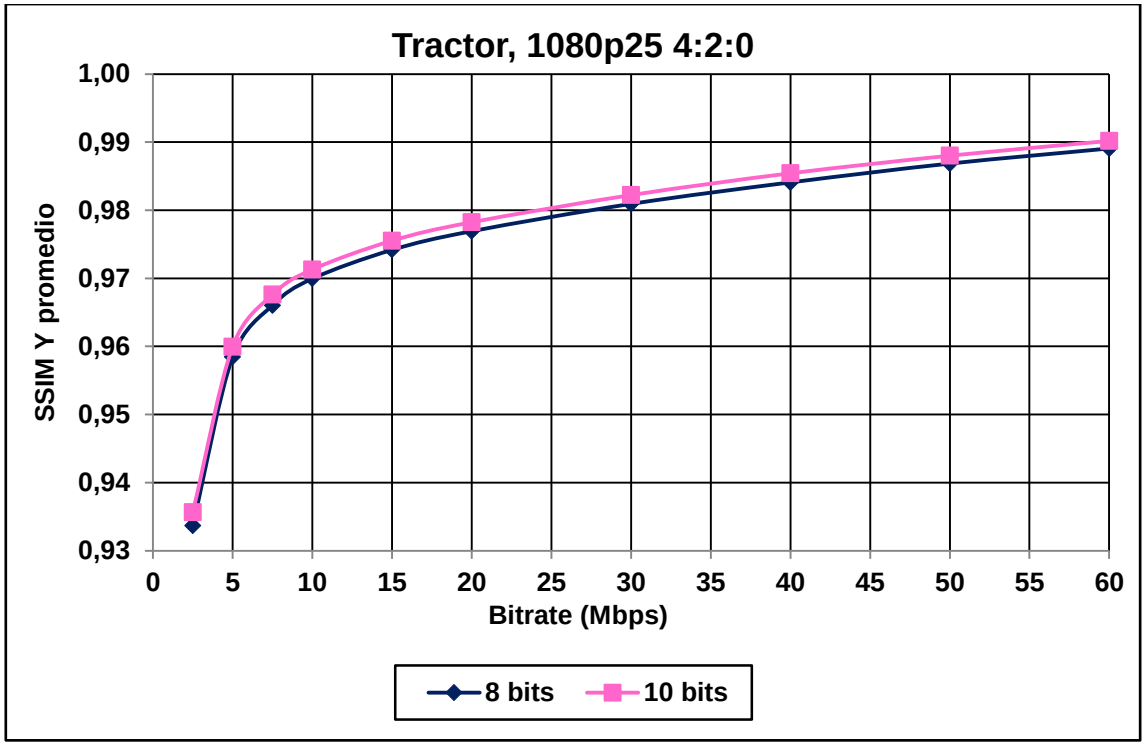

Figura 6.15 - SSIM Y promedio de la secuencia *Tractor* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 30 Mbps, en la Figura 6.16.

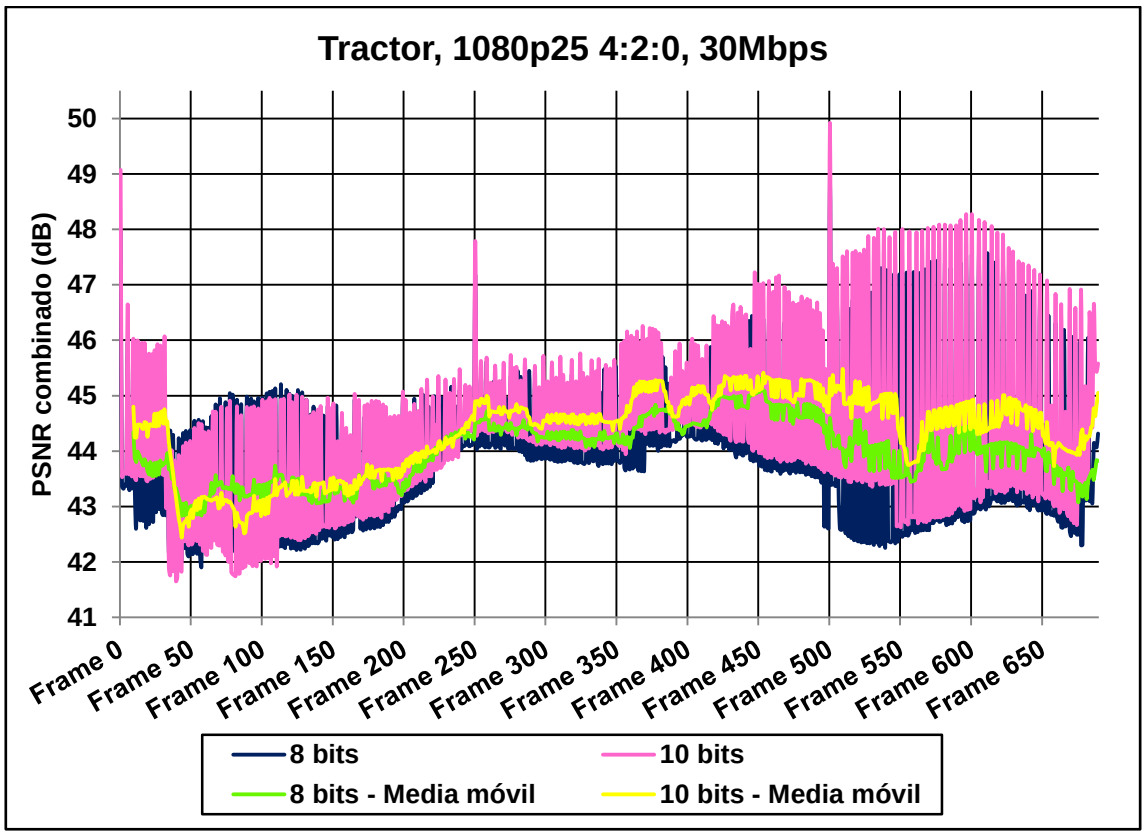

Figura 6.16 - PSNR combinado de la secuencia *Tractor*, evolución temporal

La Figura 6.14 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de aproximadamente un 8% de tasa de bits para los mismos valores de PSNR a tasas altas, no habiendo apenas diferencia para tasas bajas. Esto se traduce en un aumento de 0,3-0,4 dB de PSNR para esas tasas altas de bits.

La Figura 6.15 muestra una reducción de entre un 10% (a tasas altas) y un 5% (a tasas bajas) de tasa de bits para los mismos valores de SSIM.

Analizando la evolución temporal del PSNR en la Figura 6.16 las mayores ganancias se concentran desde aproximadamente la mitad de la secuencia hasta el final, donde la cantidad de movimiento disminuye considerablemente y se ve el cielo. En la zona con mucho movimiento no hay mejora apreciable.

La gráfica presenta esa forma de diente de sierra por la misma razón que las secuencias anteriores. En este caso x264 ha colocado tres fotogramas IDR, al principio y en los fotogramas 250 y 500 (picos máximos). El resto de picos corresponden a fotogramas tipo P y los valles a tipo B.

6.6 Secuencia *Sintel_Trailer*

Figura 6.17 - Secuencia *Sintel_Trailer*

La secuencia *Sintel_Trailer* consta de 1253 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se trata de un corto de animación por ordenador. Las escenas son variadas, con montañas nevadas, ciudades de piedra y desiertos, con una gran cantidad de movimiento en la mayoría de ellas. Los fondos y las texturas en general tienen poco detalle. La secuencia, al estar generada por ordenador, está totalmente libre de ruido. Entre las distintas escenas hay tomas completamente negras, o con texto sobre fondo negro. En la Figura 6.17 pueden verse tres capturas de distintos momentos de la secuencia.

En la Figura 6.18 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.19 se muestra el segundo gráfico, SSIM Y.

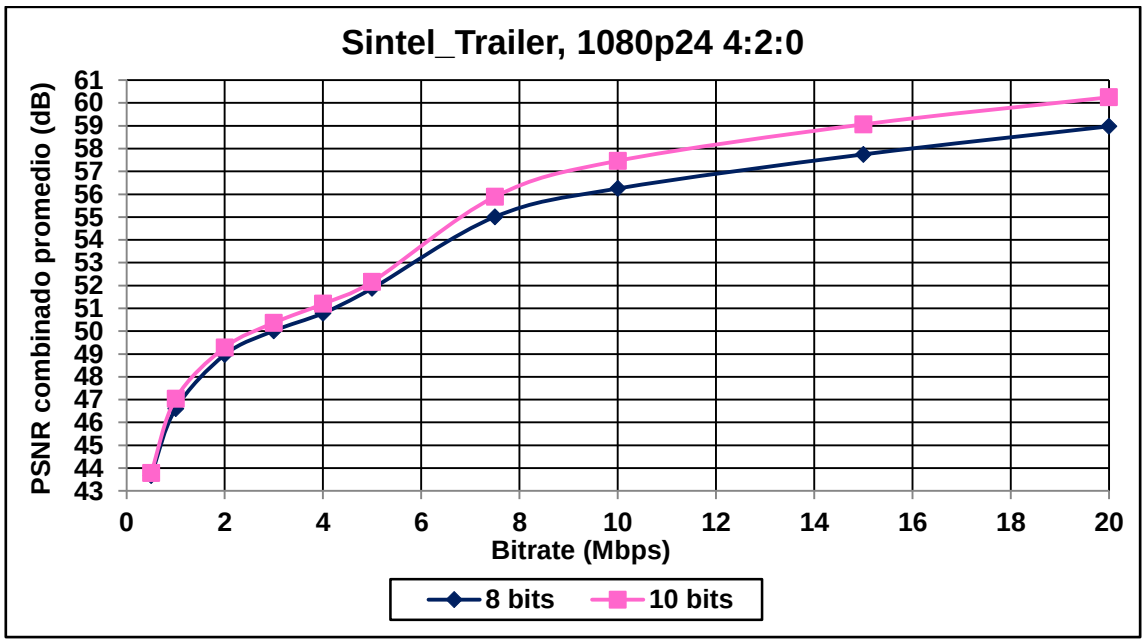

Figura 6.18 - PSNR combinado promedio de la secuencia *Sintel_Trailer* a diferentes regímenes binarios

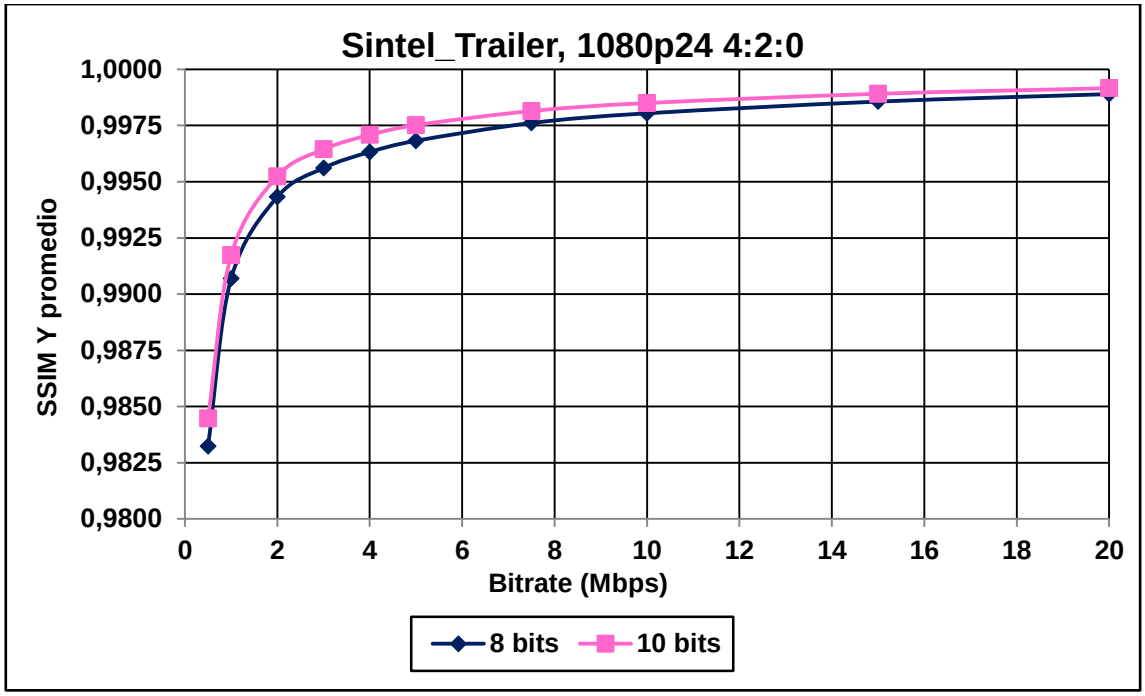

Figura 6.19 - SSIM Y promedio de la secuencia *Sintel_Trailer* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 10 Mbps, en la Figura 6.20.

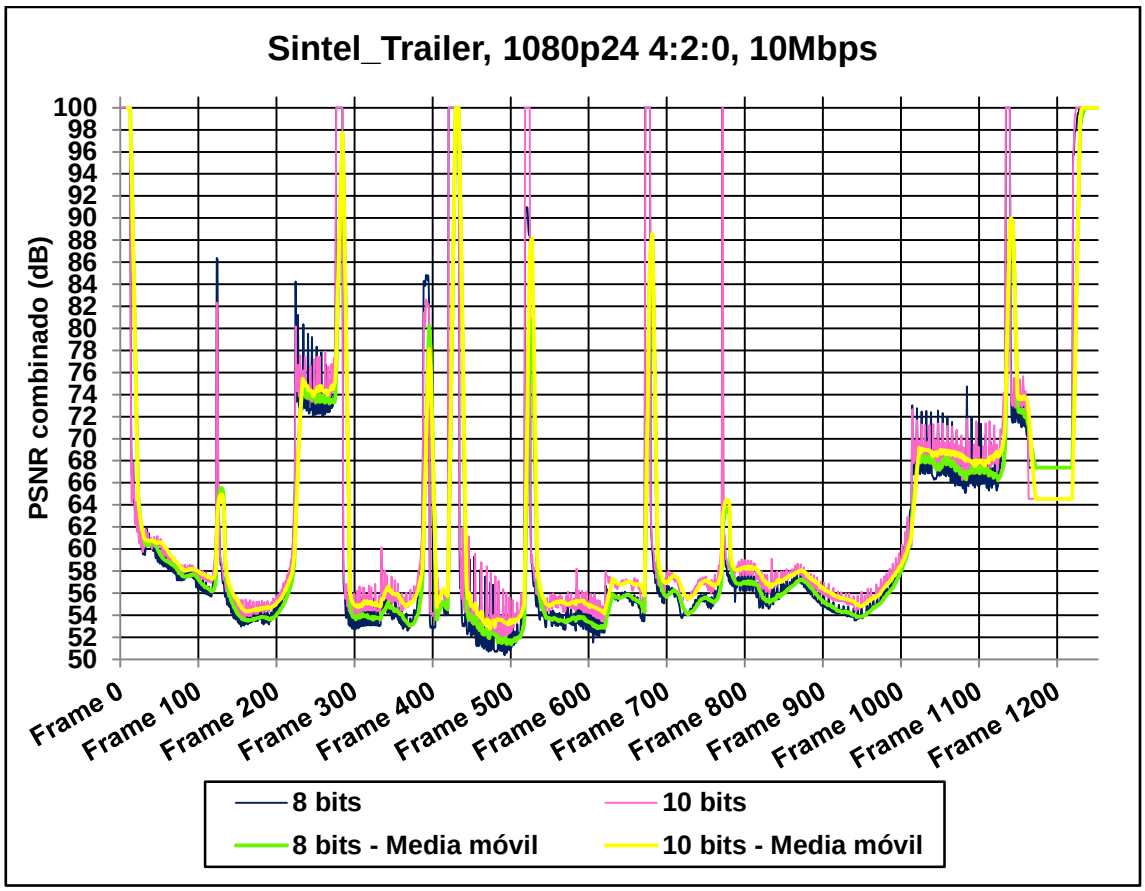

Figura 6.20 - PSNR combinado de la secuencia *Sintel_Trailer*, evolución temporal

La Figura 6.18 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de entre un 5% y un 25% de tasa de bits para los mismos valores de PSNR, o un aumento de entre 0,4 y 1,2 dB de PSNR para las mismas tasas de bits.

La Figura 6.19 muestra una reducción de entre un 30% (a tasas altas) y un 15% (a tasas bajas) de tasa de bits para los mismos valores de SSIM.

En el análisis temporal del PSNR en la Figura 6.20 cabe destacar lo dicho anteriormente, las escenas en negro se codifican tan bien que el PSNR sube hasta el máximo valor definido en el programa de análisis, 100 dB. En las escenas con texto sobre negro la situación no es tan extrema, pero si suficiente como para disparar el PSNR hasta los 70-80 dB. Estas zonas deben ignorarse, ya que no son significativas. En las partes significativas se aprecia como la ganancia de la codificación con 10 bits de precisión es casi constante y del orden de 1 dB.

6.7 Secuencia *Dinner*

Figura 6.21 - Secuencia *Dinner*

La secuencia *Dinner* consta de 950 fotogramas, con una resolución de 1920x1080 pixeles y submuestreo en los planos de diferencia de color 4:2:0.

Se trata de una secuencia totalmente sintética generada por ordenador. Representa una mesa decorada para una cena romántica desde distintos ángulos. Los fondos y las texturas en general tienen poco o muy poco detalle. Al ser generada por ordenador está totalmente libre de ruido. En la Figura 6.21 pueden verse tres capturas de distintos momentos de la secuencia.

En la Figura 6.22 se muestra el gráfico del PSNR combinado promedio de la secuencia, y en la Figura 6.23 se muestra el segundo gráfico, SSIM Y.

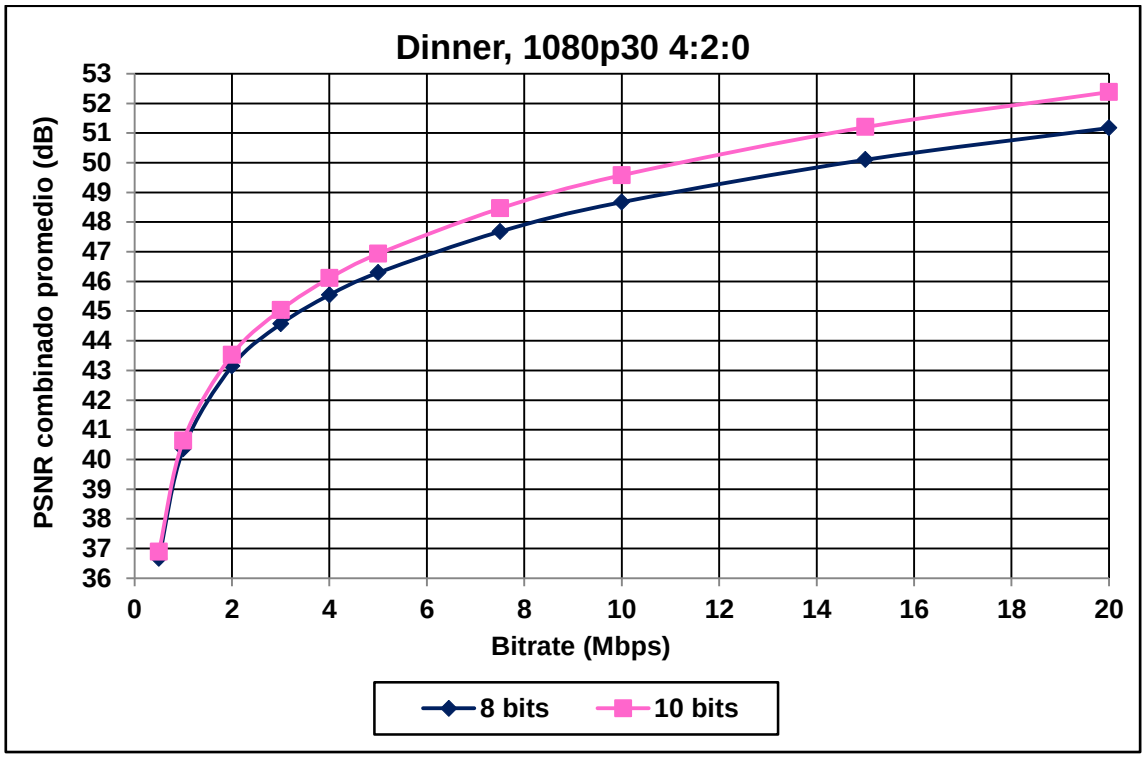

Figura 6.22 - PSNR combinado promedio de la secuencia *Dinner* a diferentes regímenes binarios

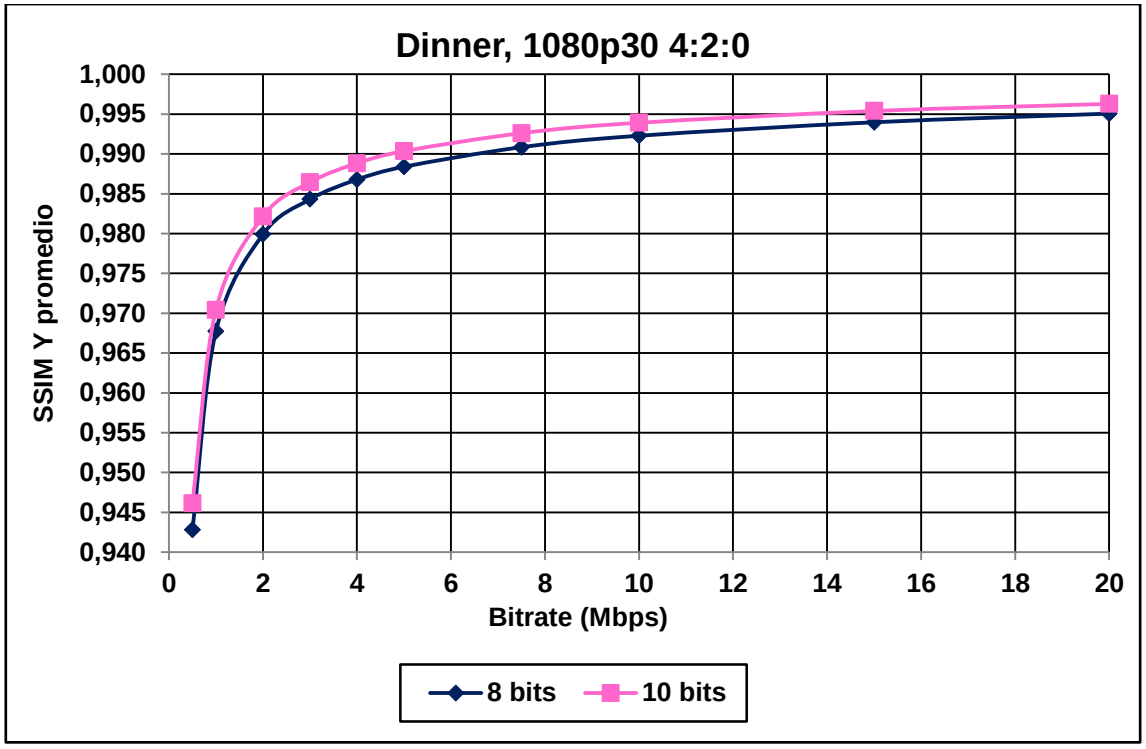

Figura 6.23 - SSIM Y promedio de la secuencia *Dinner* a diferentes regímenes binarios

Por último se muestra la evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 10 Mbps, en la Figura 6.24.

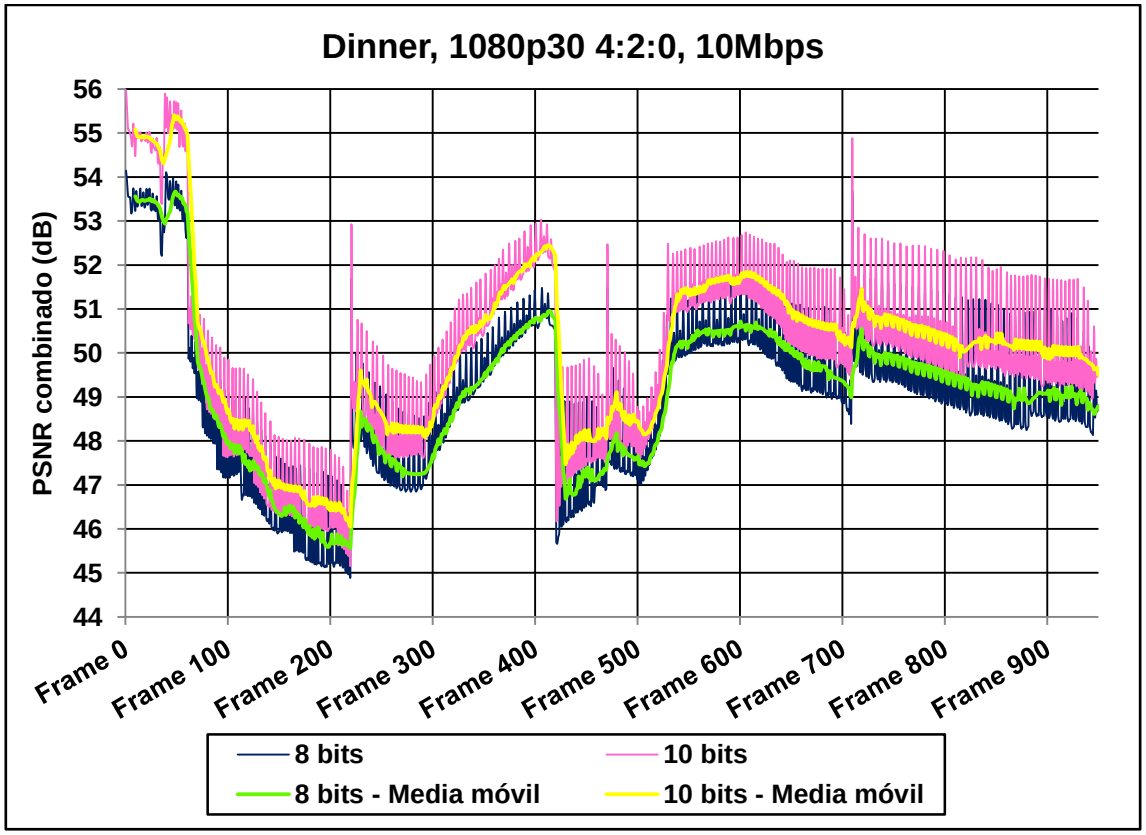

Figura 6.24 - PSNR combinado de la secuencia *Dinner*, evolución temporal

La Figura 6.22 muestra que para esta secuencia el aumento de precisión interna del codificador de 8 a 10 bits resulta en una reducción de entre un 5% y un 25% de tasa de bits para los mismos valores de PSNR, o un aumento de entre 0,3 y 1,2 dB de PSNR para las mismas tasas de bits.

La Figura 6.23 muestra una reducción de entre un 28% (a tasas altas) y un 15% (a tasas bajas) de tasa de bits para los mismos valores de SSIM.

En el análisis temporal del PSNR en la Figura 6.24 cabe destacar el impacto del aumento de precisión, con un aumento constante de aproximadamente 1 dB a lo largo de toda la secuencia.

La gráfica presenta esa forma de diente de sierra por la misma razón que las secuencias anteriores. En este caso x264 ha colocado varios fotogramas IDR, coincidiendo con los cambios de escena de la secuencia (picos máximos). El resto de picos corresponden a fotogramas tipo P y los valles a tipo B.

6.8 Secuencia *Ducks_Take_Off*

Figura 6.25 - Secuencia *Ducks_Take_Off*

La secuencia *Ducks_Take_Off* consta de 500 fotogramas. Se analizan dos resoluciones, 1280x720 y 3840x2160 con submuestreo en los planos de diferencia de color 4:2:0, y para la resolución de 1280x720 se analizan también los submuestreos en los planos de diferencia de color 4:2:2 y 4:4:4.

La toma muestra unos patos en un estanque que echan a volar. Tras iniciar el vuelo dejan tras ellos una gran cantidad de salpicaduras y ondas en el agua. Tiene una gran cantidad de detalle en las salpicaduras, lo que la hace muy difícil de codificar. La secuencia apenas tiene ruido, aunque a la resolución de 1280x720 pixeles las gotas y salpicaduras individuales son tan pequeñas que podrían considerarse ruido. En la Figura 6.25 pueden verse tres capturas de distintos momentos de la secuencia.

En primer lugar se muestran en la Figura 6.26, la Figura 6.27 y la Figura 6.28 los gráficos del PSNR combinado promedio de la secuencia, el gráfico del SSIM Y, y el gráfico de evolución del PSNR combinado a lo largo del tiempo para la tasa de bits de 10 Mbps correspondientes a la resolución de 1280x720 4:2:0.

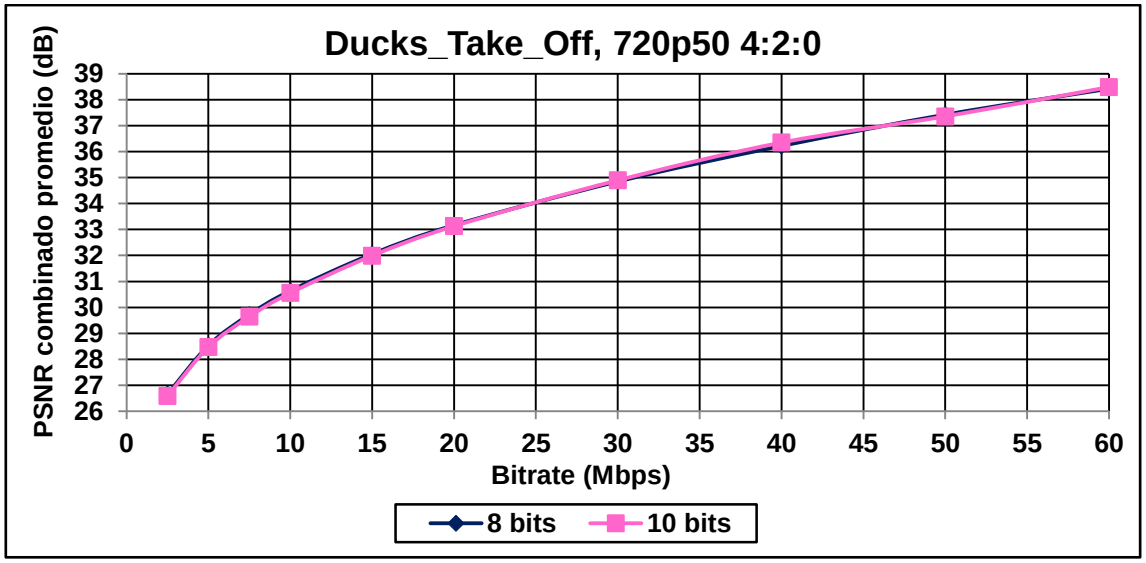

Figura 6.26 - PSNR combinado promedio de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:0

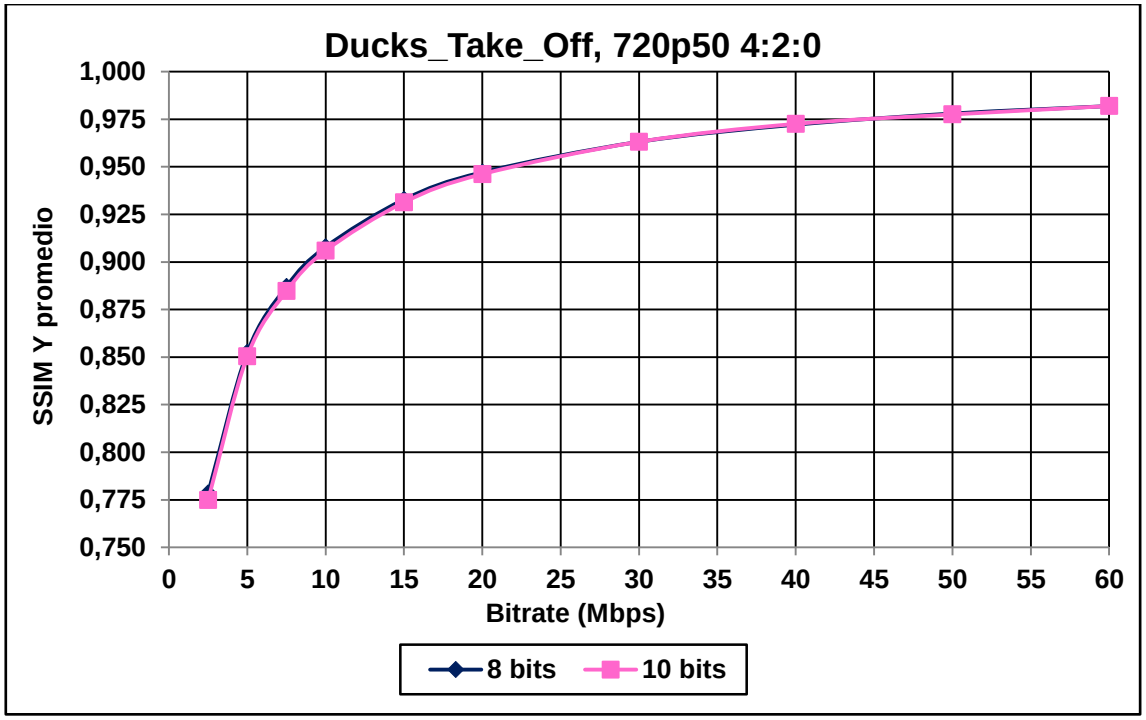

Figura 6.27 - SSIM Y promedio de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:0

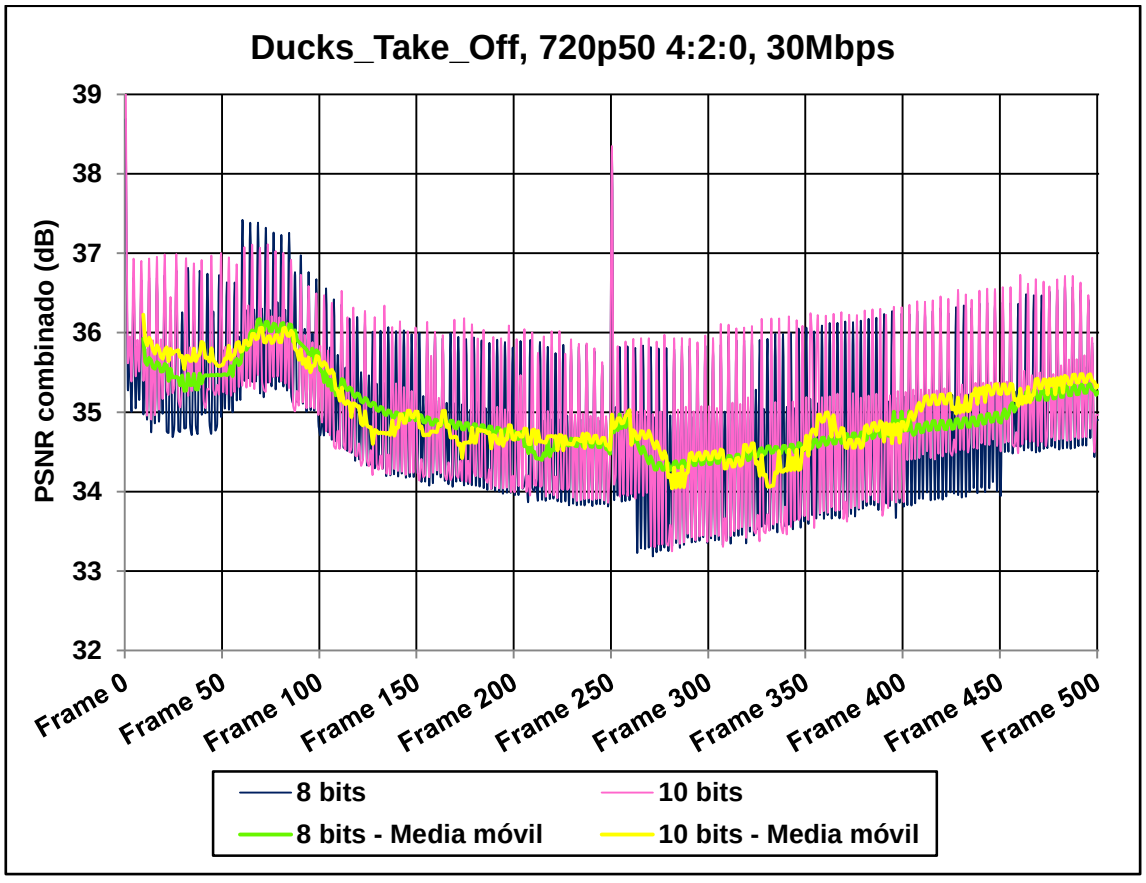

Figura 6.28 - PSNR combinado de la secuencia *Ducks_Take_Off*, evolución temporal, 1280x720 4:2:0

A continuación se muestran en la Figura 6.29 y la Figura 6.30 los gráficos del PSNR combinado promedio de la secuencia correspondientes a la resolución de 1280x720 y submuestreo de color 4:2:2 y 4:4:4.

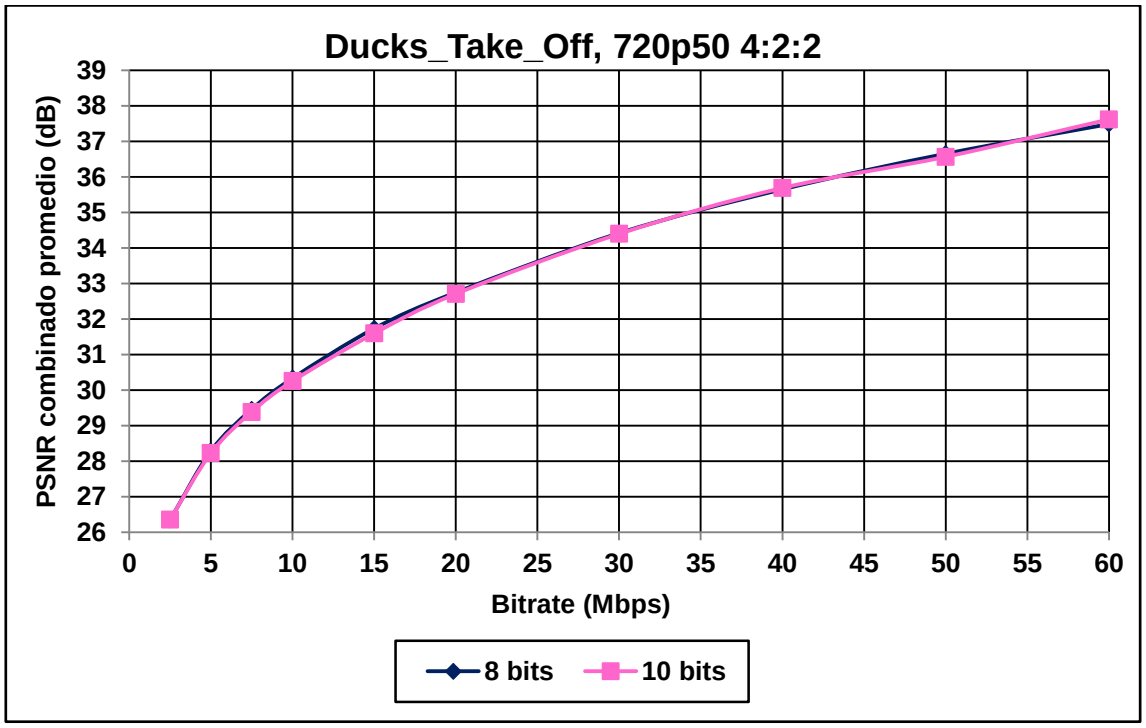

Figura 6.29 - PSNR combinado promedio de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:2

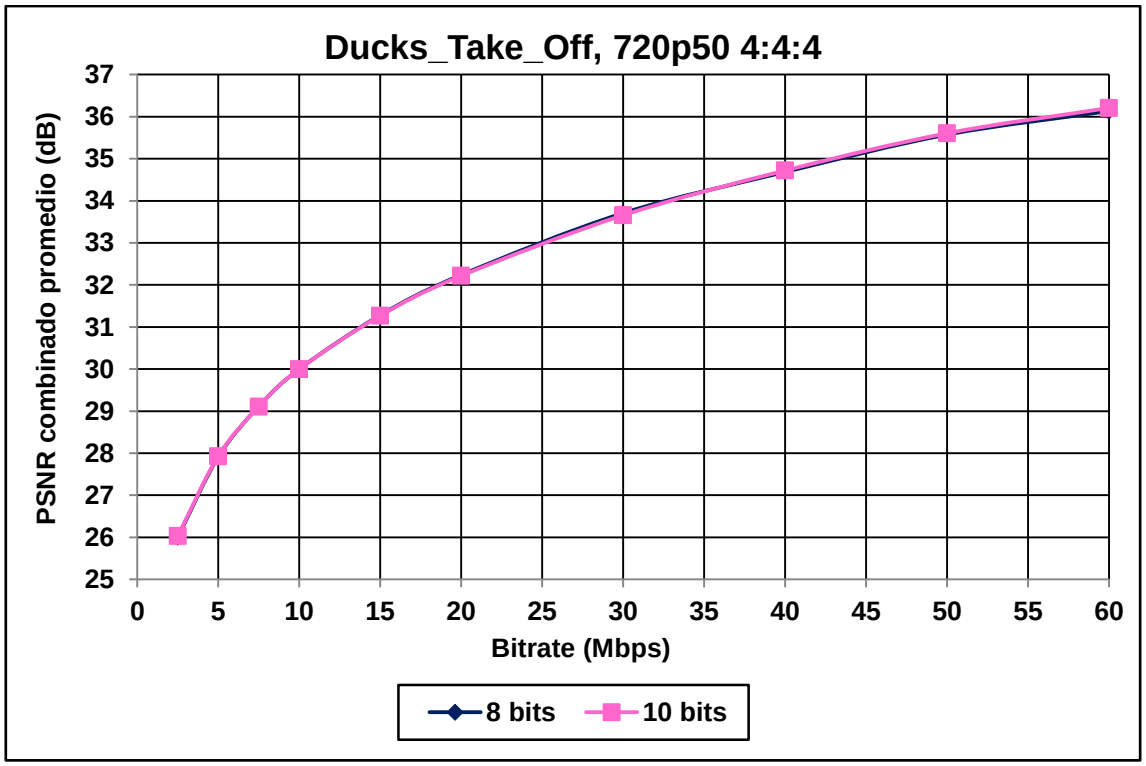

Figura 6.30 - PSNR combinado promedio de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:4:4

Por último se muestra en la Figura 6.31 el gráfico del PSNR combinado promedio de la secuencia correspondiente a la resolución de 3840x2160 y submuestreo de color 4:2:0.

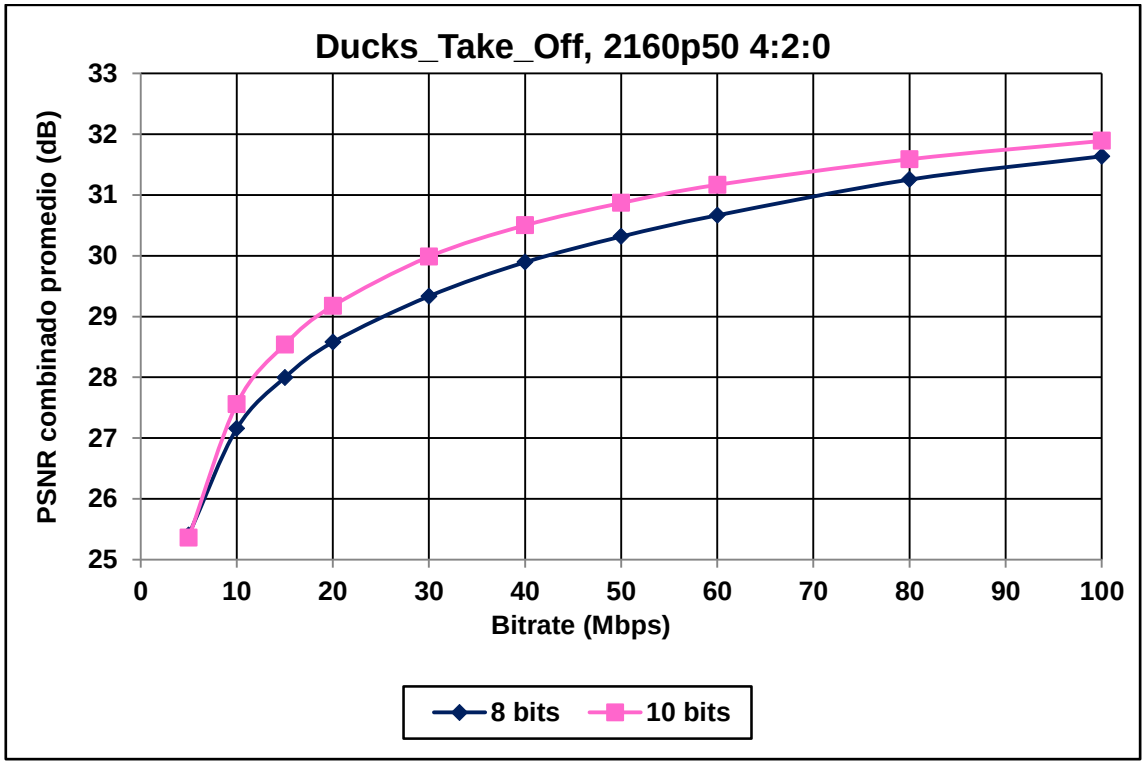

Figura 6.31 - PSNR combinado promedio de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 3840x2160 4:2:0

Los resultados muestran que para la resolución de 1280x720 pixeles en esta secuencia el aumento de precisión interna del codificador no resulta en una reducción de la tasa de bits apreciable ni para el mismo PSNR ni para el mismo SSIM.

De la misma manera el uso de diferentes tipos de submuestreo de los planos de color no presenta cambios significativos para esa resolución.

Sin embargo sí que se aprecian diferencias significativas al aumentar la resolución de la secuencia a 3840x2160. En este caso el aumento de precisión interna del codificador resulta en una media de reducción de un 25% de tasa de bits para los mismos valores de PSNR, o un aumento de entre 0,4 y 0,7 dB de PSNR para las mismas tasas de bits.

6.9 Resumen de los resultados de eficiencia de codificación

Ante los resultados del estudio de eficiencia de codificación realizado ha quedado comprobado que un aumento de la precisión interna del codificador resulta en una disminución significativa de la tasa binaria para el mismo nivel de calidad visual objetiva, incluso si el material codificado no tiene más de 8 bits de precisión inicial. Dicho aumento es muy variable dependiendo del tipo de contenido que se codifica, y está provocado por el aumento en la precisión de las predicciones que el codificador realiza durante las fases de predicción *intra*, estimación y compensación del movimiento y filtro *in-loop deblocking*, lo que consigue que el residuo a codificar como resultado de dichas predicciones sea menor, pudiendo dedicar esos bits no desperdiciados en estos puntos para aumentar la calidad general del video.

Al utilizar PSNR como medida de calidad las diferencias van desde la ausencia de mejora hasta una reducción del 25% de tasa de bits para el mismo valor de PSNR. Se aprecia que en zonas donde hay gran cantidad de ruido la ganancia es despreciable, mientras que en escenas con poco ruido o con total ausencia de él como son las secuencias por ordenador analizadas prácticamente en todo momento la codificación con 10 bits de precisión ha proporcionado una ganancia de 1 dB de PSNR sobre la codificación tradicional con 8 bits de precisión. Las ganancias también se concentran en zonas donde hay poco detalle visual, como cielos o paredes de color uniforme o formando un gradiente de color. Es conveniente resaltar que en los casos donde se ha producido mejora de calidad dicha mejora en la mayoría de los casos ha ido en aumento conforme se ha ido aumentando la tasa de bits.

Al utilizar SSIM como medida de calidad las diferencias han ido también desde la ausencia de mejora hasta reducciones del 50% de tasa binaria para la misma calidad objetiva. La distribución de las mejoras es muy similar al caso del PSNR, aunque en general la mejora de SSIM suele ser mayor en proporción a la del PSNR para el mismo régimen binario.

Del estudio también se desprende que utilizar otro tipo de submuestreo de los planos de diferencia de color distintos al 4:2:0 no provoca cambios apreciables. Si con video 4:2:0 hay mejoras para una secuencia concreta, con esa misma secuencia en formato 4:2:2 o 4:4:4 también las habrá, y viceversa.

Por último se ha dado una situación digna de especial mención al estudiar una misma secuencia a dos resoluciones distintas, 1280x720 y 3840x2160 pixeles. Mientras que en el caso de resolución 720p no ha habido apenas diferencia entre los dos tipos de

codificación en el caso de la resolución 2160p la disminución de tasa de bits para el mismo PSNR ha llegado a alcanzar el 25%.

En ningún caso de los analizados hay empeoramiento del PSNR o SSIM para una misma tasa binaria, por lo que puede decirse que en cuanto a la eficiencia de codificación el uso de más bits de precisión no presenta ninguna desventaja.

Capítulo 7 - Pruebas de rendimiento: 8 vs. 10 bits

En este capítulo se realizan las distintas pruebas de rendimiento, midiendo la velocidad tanto de codificación como de descodificación en número de fotogramas por segundo utilizando los perfiles H.264 de 8 y 10 bits de precisión. Para ello se presentan solamente los resultados obtenidos de la secuencia *Ducks_Take_Off*, ya que el análisis previo de las demás secuencias muestra que la relación de rendimiento es prácticamente constante entre los dos tipos de codificación independientemente del tipo de la secuencia analizada. Se prueban dos resoluciones, 1280x720 y 3840x2160, y para la resolución de 1280x720 se analizan los tipos de submuestreo de planos de diferencia de color 4:2:0, 4:2:2 y 4:4:4.

7.1 Configuración de las pruebas

Para estas pruebas se ha decidido codificar con parámetros tales que produzcan *bitstreams* compatibles con el estándar Blu-ray, para ofrecer una visión realista del uso de la codificación con 10 bits. Estos parámetros difieren de los utilizados en las pruebas de eficiencia, donde a costa de la velocidad de codificación (en la mayoría de los casos inferior a 1 fotograma por segundo) se conseguía la calidad máxima obtenible por x264. Dado que en estas pruebas no se pretende analizar la relación de eficiencia sino de rendimiento, y teniendo en cuenta que los parámetros de máxima eficiencia consiguen velocidades de codificación demasiado bajas para ser significativas en aplicaciones del mundo real, mediante el uso de *presets* más rápidos se han conseguido unas velocidades de codificación aceptables.

Además, los *bitstreams* resultantes no son totalmente compatibles con Blu-ray, ya que el estándar está limitado a resoluciones de hasta 1920x1080 pixeles y el tipo de submuestreo de planos de diferencia de color 4:2:0. Por ello salvo por la resolución y el tipo de submuestreo el resto de los parámetros cumplen con lo especificado en el estándar, como se detalla en [29].

Para la realización de las pruebas se han dado los siguientes pasos:

1) Para la obtención de la velocidad de codificación se codifica la secuencia con x264 a archivo *.mp4*. Una vez terminada la codificación x264 muestra por la consola la velocidad media de codificación en fotogramas por segundo (ver la

Figura 4.4). Ese valor es el que se ha representado en los distintos gráficos de resultados. Resaltar que al igual que para las pruebas de eficiencia de codificación para conseguir los ficheros codificados de 8 y 10 bits lo único que cambia es el ejecutable de x264, como se explica en el capítulo 4.1.

Todos los parámetros son idénticos en ambos casos. Como se pretende medir el rendimiento a distintas tasas binarias medias se utiliza el modo de codificación a dos pasadas, que es el único que garantiza que al final de las codificaciones se obtengan las tasas binarias medias especificadas (ver Parámetros relacionados con el control de la tasa de bits en el ANEXO II). Por tanto las líneas de comandos pasadas al codificador x264 para las distintas pruebas son:

x264 <secuencia>.y4m -o <secuencia>.mp4 --preset slower --tune film --bluray-compat --vbv-maxtare 40000 --vbv-bufsize 30000 --keyint 50 --slices 4 --bitrate ** --pass 1**

x264 <secuencia>.y4m -o <secuencia>.mp4 --preset slower --tune film --bluray-compat --vbv-maxtare 40000 --vbv-bufsize 30000 --keyint 50 --slices 4 --bitrate ** --pass 2**

En caso de estar codificando video 4:2:2 o 4:4:4 se añaden los parámetros **--input-csp i422 --output-csp i422** ó **--input-csp i444 --output-csp i444**.

2) Con los ficheros codificados obtenidos del paso anterior se realiza a continuación la prueba de rendimiento de descodificación. Para ello se han utilizado GraphStudioNext y LAV Filters, como se explica en el capítulo 4.6.

Los resultados obtenidos se representan de la siguiente manera:

- Para las pruebas de codificación, un gráfico por cada resolución y tipo de submuestreo mostrando el número de fotogramas por segundo obtenidos en cada una de las dos pasadas, codificando con 8 y 10 bits.
- Para las pruebas de descodificación, un gráfico por cada resolución y tipo de submuestreo mostrando el número de fotogramas por segundo obtenidos, con los ficheros de 8 y 10 bits.

Todas las pruebas se han realizado en un equipo con una CPU Intel Core i5-2500K, con frecuencia de trabajo de 3,3 GHz, turbo de 3,7 GHz y 4 núcleos, junto con 8 GB de memoria RAM DDR3 a 1600MHz. Tanto x264 como LAV Filters utilizan los 4 núcleos al máximo durante toda la duración de los tests.

7.2 Codificación

En las siguientes figuras se muestran los resultados de codificación obtenidos.

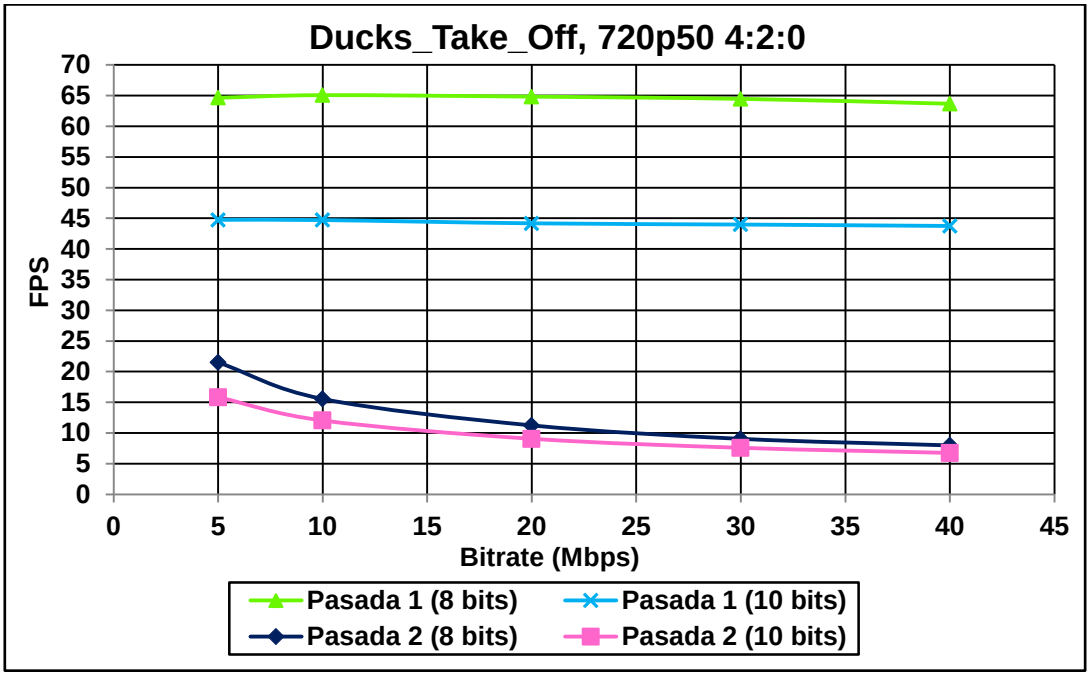

Figura 7.1 - Velocidad de codificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:0

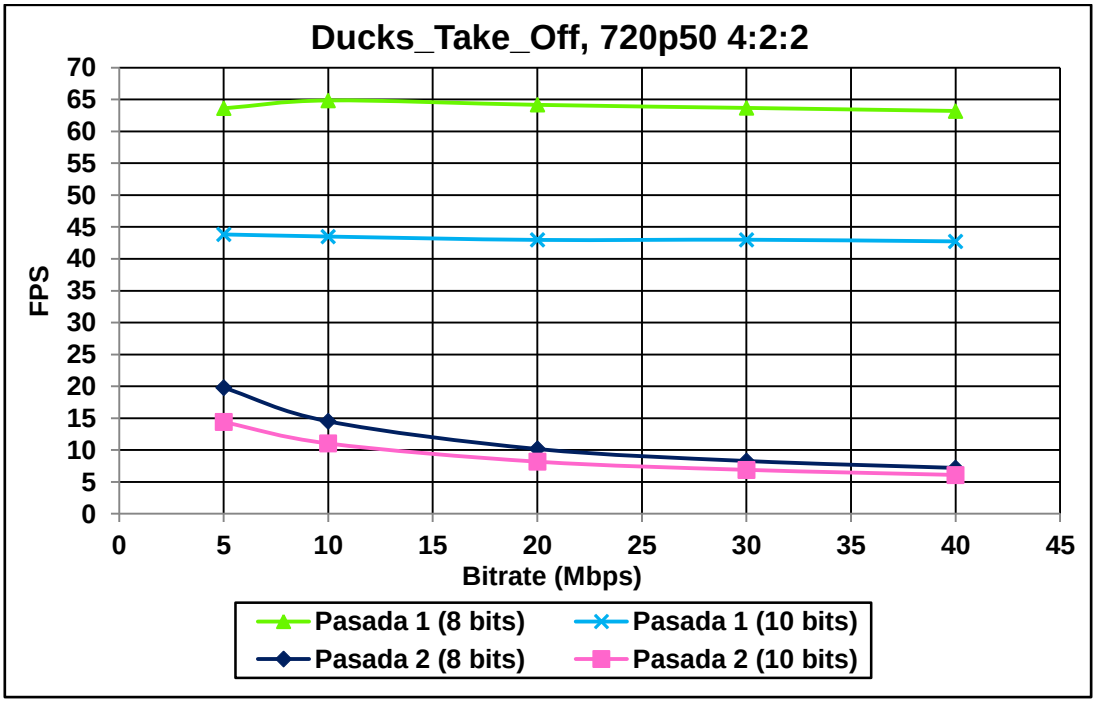

Figura 7.2 - Velocidad de codificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:2

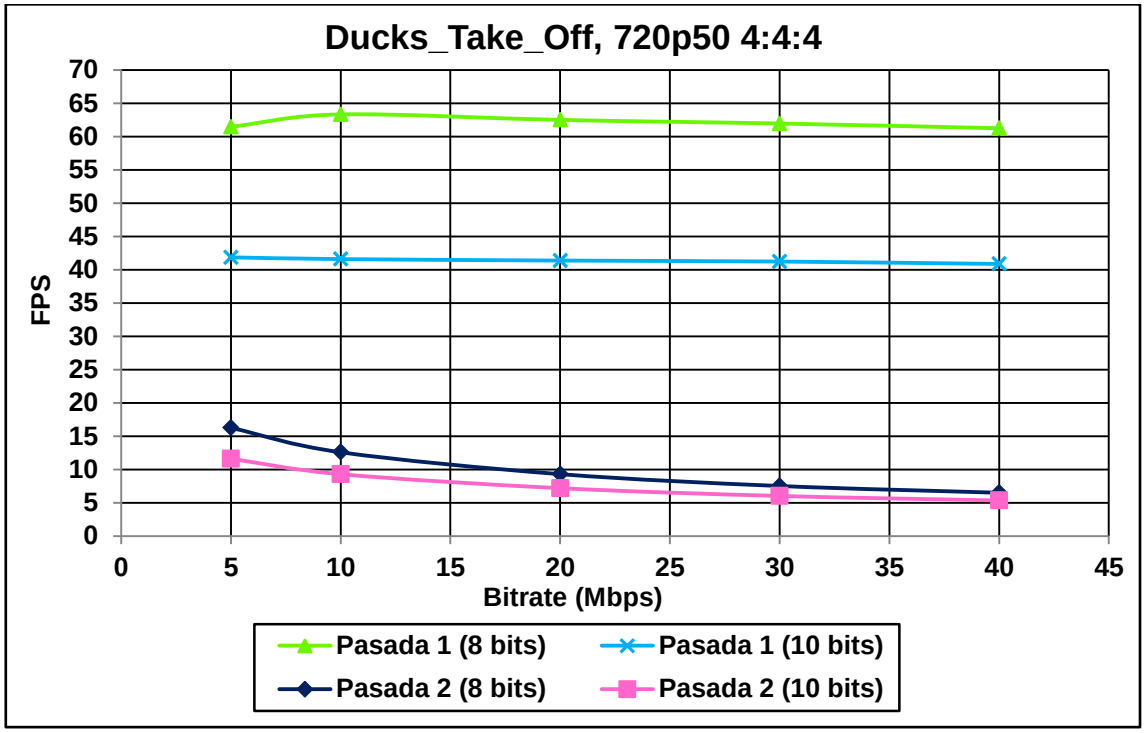

Figura 7.3 - Velocidad de codificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:4:4

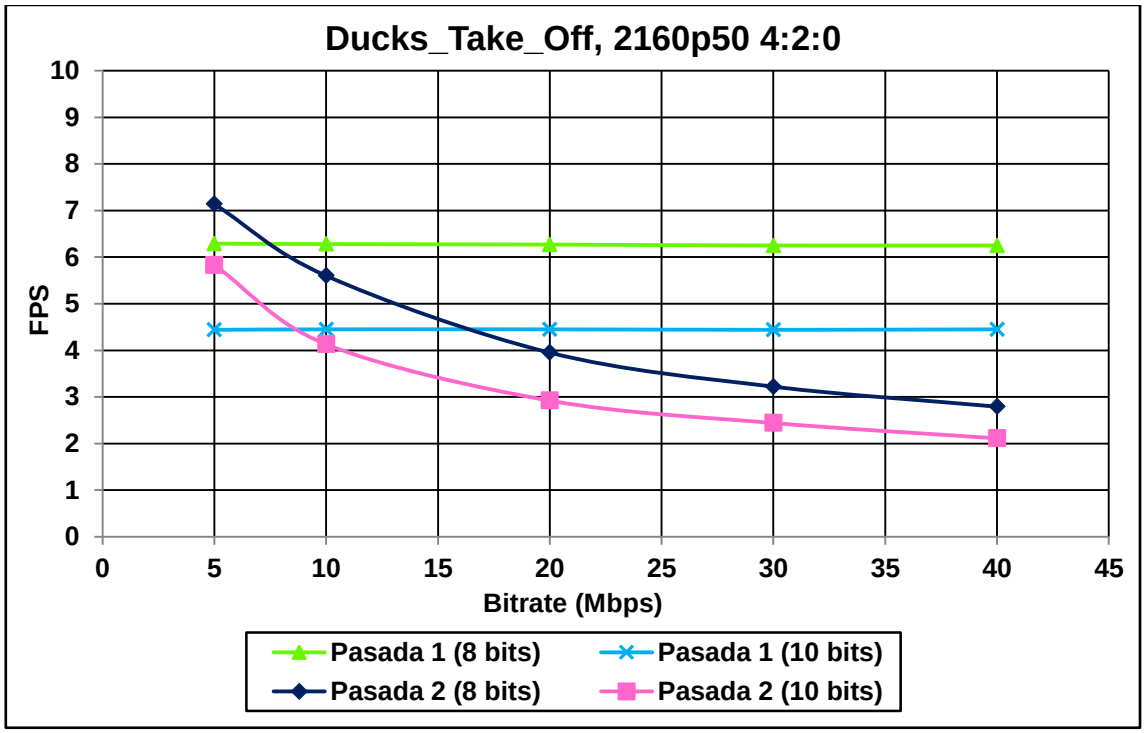

Figura 7.4 - Velocidad de codificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 3840x2160 4:2:0

Los resultados muestran una disminución del orden de 30-35% del rendimiento en la primera pasada al pasar de 8 a 10 bits de precisión, prácticamente constante con tasa de bits, resolución y diferentes tipos de submuestreo.

En la segunda pasada la disminución es más variable, dependiente sobre todo de la tasa de bits. Para tasas bajas la disminución es de aproximadamente un 25%, mientras que para tasas altas se reduce hasta un 15% a la resolución de 1280x720 pixeles. Los distintos tipos de submuestreo no difieren apenas entre ellos. Para la resolución de 3840x2160 la reducción es más o menos constante, alrededor de un 25%.

7.3 Descodificación

En las siguientes figuras se muestran los resultados de descodificación obtenidos.

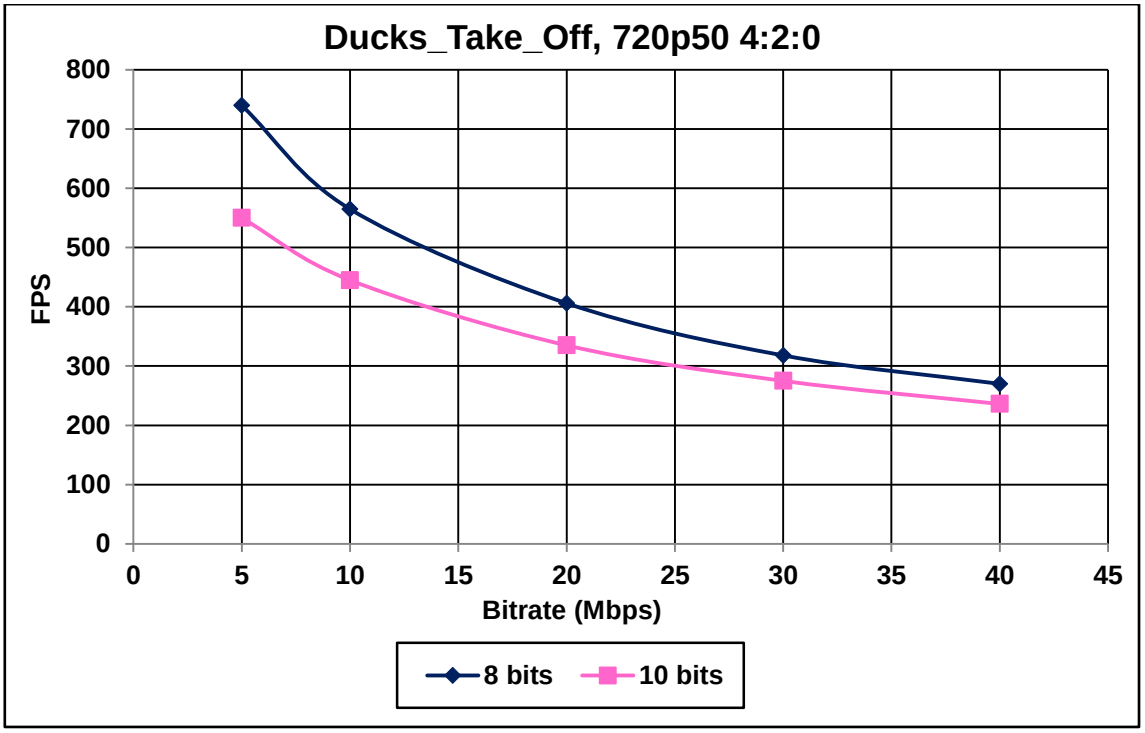

Figura 7.5 - Velocidad de descodificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:0

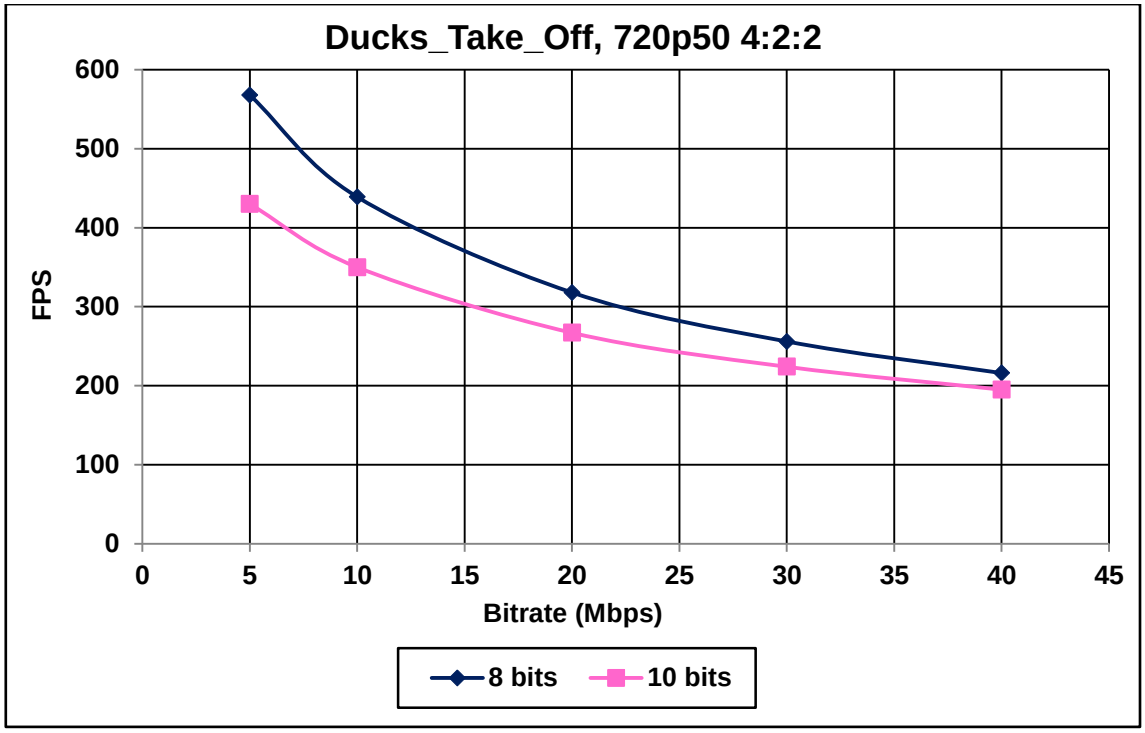

Figura 7.6 - Velocidad de descodificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:2:2

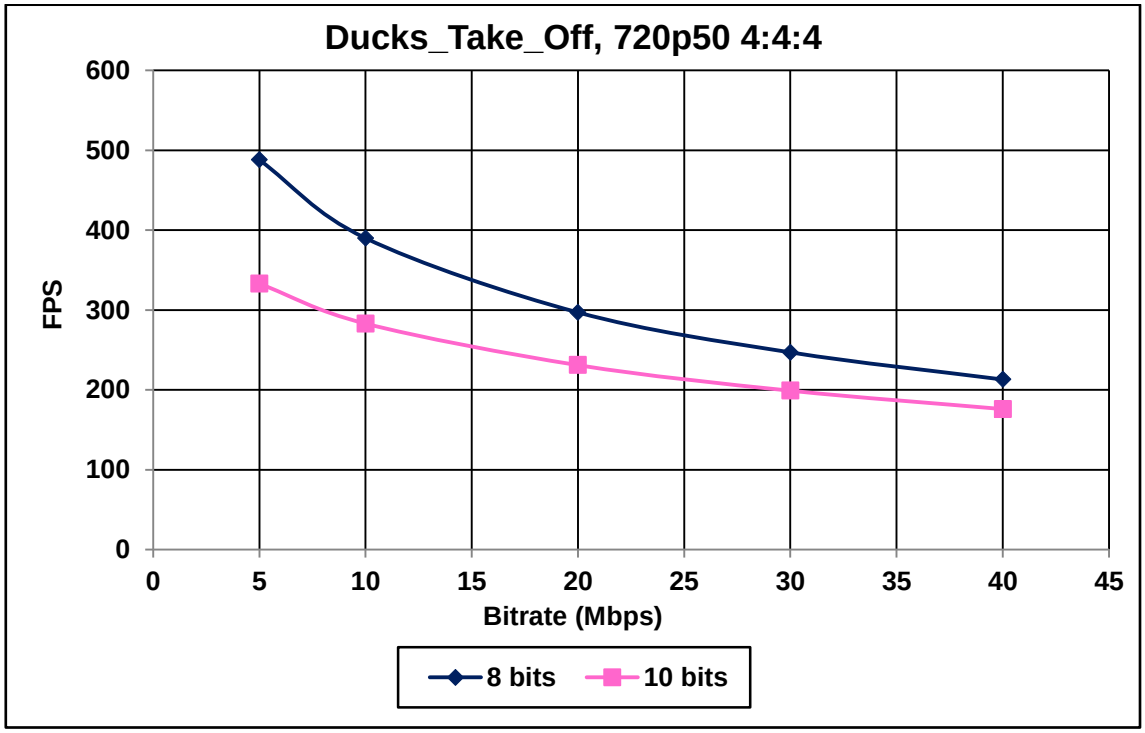

Figura 7.7 - Velocidad de descodificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 1280x720 4:4:4

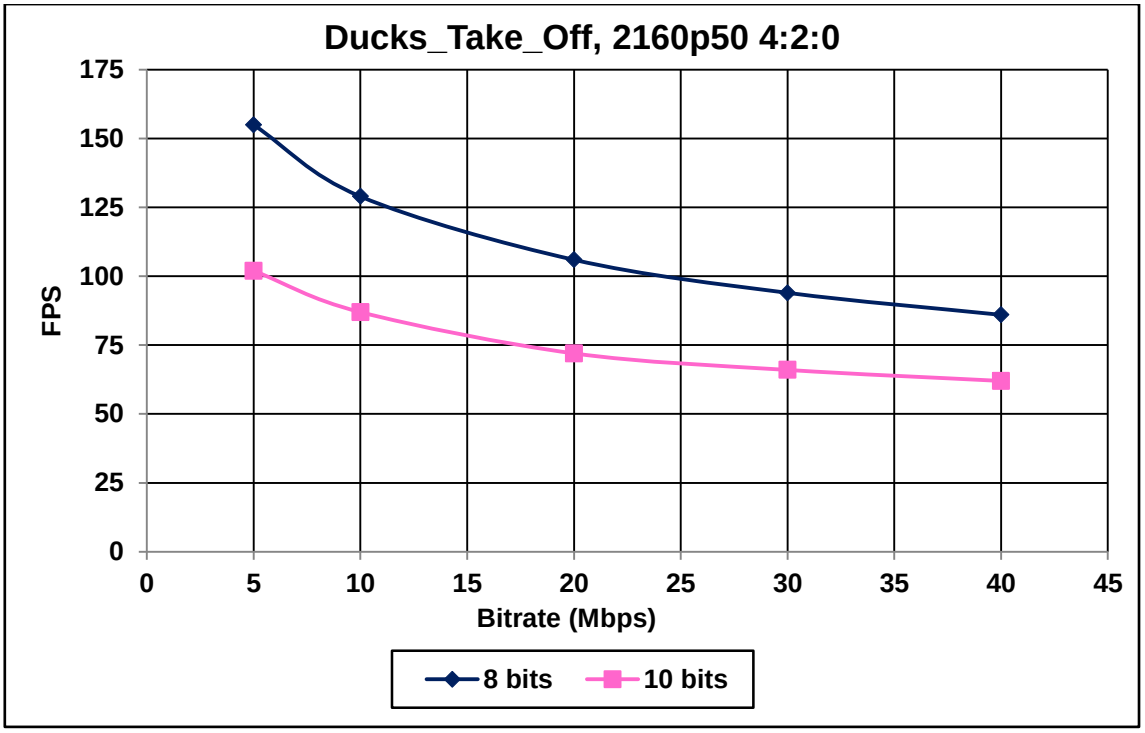

Figura 7.8 - Velocidad de descodificación de la secuencia *Ducks_Take_Off* a diferentes regímenes binarios, 3840x2160 4:2:0

Los resultados muestran, para la resolución de 1280x720 pixeles, una disminución variable del rendimiento según la tasa de bits: del orden de 25% para tasas bajas y del orden de un 10% para tasas altas al pasar de 8 a 10 bits de precisión. Los mismos rangos son aplicables para todos los tipos de submuestreo de color.

En cuanto a la resolución de 3840x2160 la disminución del rendimiento se acentúa hasta un 35% en tasas bajas y un 30% en tasas altas.

Capítulo 8 - Conclusiones y líneas futuras de trabajo

Una vez finalizado el desarrollo del proyecto, se analiza el trabajo realizado para extraer las conclusiones del mismo en el apartado de conclusiones. A continuación se citan unas posibles líneas futuras de trabajo, que han ido surgiendo durante la realización de los distintos capítulos.

8.1 Conclusiones

En primer lugar, el desarrollo del programa de análisis se ha completado con éxito, ya que realiza todas las funciones que se marcaron como objetivos. Ha sido elemento indispensable para poder realizar el estudio de eficiencia de codificación. Aunque la aplicación se ha utilizado con video de hasta 10 bits por muestra, tests experimentales demuestran que funciona correctamente con video de 16 bits por muestra como era requerido.

A continuación se ha abordado el objetivo principal del proyecto, estudiar la eficiencia de codificación utilizando 8 y 10 bits por muestra. En el estudio realizado ha quedado comprobado que un aumento de la precisión interna del codificador resulta en una disminución significativa de la tasa binaria para el mismo nivel de calidad visual objetiva, incluso si el material codificado no tiene más de 8 bits de precisión inicial. Dicha disminución es muy variable dependiendo del tipo de contenido que se codifica, y está provocado por el aumento en la precisión de las predicciones que el codificador realiza durante las fases de predicción *intra*, estimación y compensación del movimiento y filtro *in-loop deblocking*. Esta mejora en las predicciones consigue que el residuo a codificar sea menor, pudiendo dedicar esos bits no desperdiciados en estos puntos a aumentar la calidad general del video.

Gracias al estudio de la evolución temporal de la calidad se ha podido determinar que las ganancias se concentran en zonas donde hay poco detalle visual, como cielos y paredes de color uniforme o formando un gradiente de color. La presencia de ruido es determinante, ya que las mayores mejoras se consiguen en escenas libres de ruido. Esto concuerda con que la mejora sea mayor en secuencias con poco detalle, ya que el ruido a efectos de codificación es detalle muy fino en sí mismo. Es conveniente resaltar que en todos los casos donde se ha producido mejora de calidad dicha mejora ha ido en

aumento conforme se ha ido aumentando la tasa de bits. En ningún caso de los analizados ha habido empeoramiento de la calidad para una misma tasa binaria, el uso de más bits de precisión no presenta ninguna desventaja en este sentido.

Respecto al tipo de submuestreo de los planos de diferencia de color, el uso de otros tipos distintos al 4:2:0 no provoca cambios apreciables.

Sin embargo, y entrando en el último objetivo del proyecto, todas estas mejoras tienen un coste, y ese coste se presenta en forma de disminución moderada del rendimiento tanto de codificación como de descodificación.

Hay que tener en cuenta que los resultados de este proyecto se han alcanzado utilizando soluciones por software corriendo en hardware de propósito general, no en soluciones por hardware altamente especializadas. La gran mayoría de dispositivos de consumo dependen de estas soluciones hardware para codificar y decodificar video, y es requisito indispensable el desarrollo de dichas soluciones para conseguir que la tecnología haga acto de presencia entre las masas, ya que en la actualidad el uso de los perfiles H.264 de más de 8 bits de precisión está prácticamente restringido al mundo profesional [30] y a un grupo muy reducido de aplicaciones dentro del mundo PC, como son las utilizadas en este proyecto.

Sin embargo el uso de esta tecnología en el futuro parece estar garantizado. El sucesor del estándar H.264, el estándar H.265 HEVC (*High Efficiency Video Coding*), actualmente en desarrollo por el grupo MPEG y la UIT, utilizará una precisión interna 4 bits mayor que la del video a codificar [31]. El uso masivo de la codificación de más de 8 bits de precisión interna comenzará una vez se apruebe la primera versión del nuevo estándar a lo largo de 2013 y H.265 vaya ocupando progresivamente el lugar que ocupa hoy día H.264.

8.2 Líneas futuras de trabajo

Se proponen los siguientes trabajos futuros, surgidos a lo largo de la ejecución del proyecto:

- Estudio detallado de las mejoras en la predicción al codificar con más de 8 bits, viendo en qué partes del proceso de codificación ocurren. Requiere entrar en profundidad en el código de x264 o un codificador similar.
- Estudio detallado del impacto de codificar con más de 8 bits en la velocidad de codificación y descodificación. Identificar qué partes del codificador se ven más

afectadas por el aumento de precisión, y encontrar alguna optimización para mejorar el rendimiento.

- En este proyecto sólo se trabaja con 8 y 10 bits de precisión, pero el estándar admite hasta 14 bits utilizando el perfil *High 4:4:4 Predictive*. Se propone utilizar codificadores compatibles con dicho perfil y observar la progresión de eficiencia de codificación desde 8 hasta 14 bits.
- Estudio de algún codificador/descodificador por hardware con soporte para más de 8 bits de precisión.
- Estudio de la codificación con más de 8 bits en el futuro estándar H.265/HEVC.

ANEXO I

Parámetros del programa de análisis

En este anexo se detallan los parámetros que acepta el programa de análisis de secuencias de video sin comprimir creado como parte de este proyecto.

-o, -orig, -original (cadena de caracteres)

Ruta relativa al fichero original. Requerido.

-c, -comp, -compressed (cadena de caracteres)

Ruta relativa al fichero comprimido. Requerido.

-r, -res, -resolution (enteroxentero)

Resolución de los dos ficheros. Ambas resoluciones deben ser iguales. Requerido si los ficheros de entrada son en formato YUV. Formato: anchoxalto.

-csp, -colorspace (cadena de caracteres)

Tipo de submuestreo de los planos de diferencia de color de los dos ficheros de entrada. Debe ser igual en ambos ficheros. El tipo puede ser yuv420, yuv422 ó yuv444. Valor por defecto: yuv420.

-ob, -orig_bit_depth (entero)

Número de bits por muestra del fichero original. Valor por defecto: 8.

-cb, -comp_bit_depth (entero)

Número de bits por muestra del fichero comprimido. Valor por defecto: 8.

-csv_output_file (cadena de caracteres)

Ruta relativa al fichero de resultados .csv. Si este parámetro no está presente el programa no exportará los resultados en dicho formato.

-csv_decimal_separator (cadena de caracteres)

Separador decimal usado el el fichero .csv. Usar "comma" o "," en países europeos y “dot" o "." en América. Valor por defecto: “.”

-no_results

El programa no escribirá los resultados en la consola.

ANEXO II

Parámetros de codificación x264

Parámetros relacionados con los datos de entrada/salida

- **-o, --output**

  Especifica el fichero de salida. Dependiendo de la extensión del archivo que se introduzca el *bitstream* resultante de la codificación se almacenará en un contenedor u otro. Estos contenedores pueden ser: .264 (*bitstream* en bruto), .mp4 (estándar MPEG-4 Part 14), .mkv (Matroska, [32]) o .flv (Flash Video, [33]).

- **--input-csp**

  Especifica el tipo de submuestreo de crominancia utilizado en el video de origen. Opciones válidas: i420, i422, i444 o rgb.

  Valor por defecto: i420.

- **--output-csp**

  Especifica el tipo de submuestreo de crominancia que se desea que tenga el video de salida. Lo normal es que no se requiera una conversión entre entrada y salida, por lo que el valor de este parámetro y el de **--input-csp** deben coincidir en ese caso. Opciones válidas: i420, i422, i444 o rgb.

  Valor por defecto: i420.

- **--input-depth**

  Especifica el número de bits por muestra del video de origen. Opciones válidas: de 8 a 16.

  Valor por defecto: 8.

- **--input-res**

  Especifica la resolución del video de origen. Útil cuando se utilizan ficheros que no proporcionan información de resolución, como el .yuv.

- **--fps**

  Especifica el número de fotogramas por segundo del video de origen y de destino.

- **--bluray-compat**

  Aunque el formato de video doméstico Blu-ray permite utilizar el estándar H.264 para codificar las pistas de video no todas las opciones que permite el estándar están permitidas. Esta opción permite producir *bitstreams* compatibles con dicho formato.

- **--psnr**

  Especifica si al final de la codificación x264 debe mostrar datos calculados de PSNR.

- **--ssim**

  Especifica si al final de la codificación x264 debe mostrar datos calculados de SSIM.

- **--threads**

  Especifica el número de hilos a utilizar durante la codificación. Un mayor número de hilos resulta en una velocidad de codificación mayor.

  Valor por defecto: 1,5 x nº de núcleos disponibles

Parámetros relacionados con decisión de tipo de fotograma

- **-I, --keyint**

  Controla el tamaño máximo de los GOPs (*Groups Of Pictures*, grupos de imágenes), o lo que es lo mismo, controla el número máximo de fotogramas de cualquier tipo que puede haber entre dos fotogramas IDR. Este parámetro es muy importante, ya que cuanto mayor sea el número de fotogramas IDR menor será el ratio de compresión, por necesitar estos fotogramas muchos más bits para codificarse que los fotogramas P y B. Sin embargo un mayor número de fotogramas IDR favorece el acceso aleatorio a diferentes partes del vídeo y también facilita la posible futura edición del vídeo, ya que por definición solo a partir de estos fotogramas se puede empezar a decodificar con garantías de no encontrarse fotogramas P o B corruptos. Estándares como el Blu-ray obligan a colocar un fotograma IDR por segundo.

  Valor por defecto: 250

- **-i, --min-keyint**

  Controla el tamaño mínimo de los GOPs, o lo que es lo mismo, controla el número mínimo de fotogramas de cualquier tipo que puede haber entre dos fotogramas IDR.

  Valor por defecto: el valor mínimo entre (**--keyint** / 10) y **--fps**

- **-b, --bframes**

  Controla el máximo número de fotogramas tipo B que pueden colocarse consecutivamente entre fotogramas I y P. Este parámetro es muy importante, ya que cuanto mayor sea el número de fotogramas B mayor será el ratio de compresión, por necesitar menos bits para codificarse que los fotogramas I o P. El rango válido de valores va desde 0 (no hay ningún fotograma B) a 16.

  Valor por defecto: 3

- **--b-adapt**

  Controla el algoritmo que usa x264 para decidir si para un fotograma determinado conviene colocar uno tipo P o uno tipo B. Existen 3 opciones:

  - 0: Siempre colocar un tipo B.
  - 1: Algoritmo rápido, decisiones pobres. Solo conviene utilizar esta opción cuando se utiliza también **--bframes 16**.
  - 2: Algoritmo lento, decisiones más acertadas. Más lento cuanto mayor número de fotogramas B consecutivos pueda haber.

  Valor por defecto: 1

- **--b-pyramid**

  Controla el uso de fotogramas B como referencias de otros fotogramas. Existen 3 opciones:

  - none: ningún fotograma B es usado como referencia.
  - strict: permite que solo un fotograma B pueda ser utilizado como referencia en cada miniGOP.
  - normal: permite que cualquier número de fotogramas B puedan ser utilizados como referencias en cada miniGOP.

  Valor por defecto: normal

- **-r, --ref**

  Controla el tamaño del DPB (*Decoded Picture Buffer*, memoria de imágenes decodificadas), o lo que es lo mismo, controla el número de fotogramas anteriores que cada fotograma P puede utilizar como referencias. Este parámetro afecta al ratio de compresión, ya que el número de bits que se necesitan para codificar un fotograma P suele ser menor cuantas más imágenes de referencia se utilicen para codificarlo. El rango válido va desde 1 hasta 16.

  Valor por defecto: 3

- **-f, --deblock (alfa:beta)**

  Controla el filtro in-loop deblocking que es parte del estándar. El parámetro alfa controla la intensidad del filtro: valores mayores aplican un filtro más fuerte y viceversa. El parámetro beta controla el criterio a seguir a la hora de aplicar o no el filtro: valores mayores hacen que el bloque a filtrar tenga que ser más detallado para ser filtrado y viceversa.

  Valor por defecto: 0:0

- **--slices**

  Selecciona el número de *slices* a utilizar en cada fotograma. Utilizar *slices* disminuye la eficiencia de codificación, por lo que su uso debe evitarse si es posible. Por el contrario cada *slice* puede decodificarse independientemente de las demás, creándose así una manera fácil de paralelizar el proceso de descodificación, lo que es muy interesante para procesadores multinúcleo. El estándar Blu-ray obliga a que cada fotograma esté codificado con 4 *slices*.

  Valor por defecto: 1

- **--tff, --bff**

  Activan el modo de codificación entrelazado MBAFF. El orden de campo se indica con **--tff** (*top field first*, campo superior primero) o **--bff** (*bottom field first*, campo inferior primero).

Parámetros relacionados con el control de la tasa de bits

- **-q, --qp**

  El primero de los 3 modos de control de tasa de bits que posee x264. En él, x264 codifica en modo cuantificador constante (*Constant Quantizer*). Por definición este modo no es compatible con la cuantificación adaptativa, ya que fuerza un cuantificador determinado en todo el fotograma. La tasa de bits del video codificado resultante no es conocida, ya que, para el mismo cuantificador, una escena con mucho detalle resultará en más bits que una con poco detalle. **--qp 0** activa el modo de codificación sin pérdidas *lossless*.

- **-B, --bitrate**

  El segundo de los 3 modos de control de tasa de bits que posee x264. En él, x264 codifica intentando que la tasa de bits media del video sea igual a la especificada, en kilobits por segundo. En este modo el tamaño final del video codificado es conocido (tasa media de bits por segundo * nº de segundos = tamaño final), pero la calidad no lo es. Este parámetro es comúnmente usado junto con **--pass**, para realizar codificaciones a dos pasadas.

- **--crf**

  El tercero de los 3 modos de control de tasa de bits que posee x264. En él x264 intenta conseguir calidad constante durante toda la duración del vídeo. La unidad básica inventada de medida de la calidad es el *ratefactor*, de ahí el nombre del modo: *Constant RateFactor*. En este modo ni el tamaño final del video codificado ni la tasa media de bits es conocida. Para un mismo valor de **--crf**, o lo que es lo mismo, para conseguir la misma calidad en dos secuencias diferentes, x264 puede necesitar más bits en una que en otra para conseguir la misma calidad según la complejidad de éstas.

- **--vbv-maxrate**

  Controla la máxima velocidad a la que el VBV (*Video Buffering Verifier*, verificador de buffer de vídeo) puede rellenarse, o lo que es lo mismo, la máxima velocidad de bits en kilobits/s con la que puede codificarse el vídeo en cualquier momento. Esta opción solo debe utilizarse si el medio donde se va a reproducir posteriormente lo necesita, ya que limitar la tasa de bits puede hacer que la calidad se vea afectada negativamente. Por ejemplo, la especificación Blu-ray requiere, para videos con resolución 1920x1080, un valor de 40000.

  Valor por defecto: 0

- **--vbv-bufsize**

  Indica el tamaño del VBV en kilobits. Esta opción solo debe utilizarse si el medio donde se va a reproducir posteriormente lo necesita, ya que limitar la tasa de bits, lo cual sucede si el buffer es pequeño y la complejidad es elevada durante un largo periodo de tiempo, puede hacer que la calidad se vea afectada negativamente. Por ejemplo, la especificación Blu-ray requiere, para videos con resolución 1920x1080, un valor de 30000.

  Valor por defecto: 0

- **--chroma-qp-offset**

  Añade un offset al cuantificador de crominancia comparado con el de luminancia. Debido a que el ojo humano es menos sensible a la calidad de los planos de diferencia de color, este parámetro se puede utilizar para darle más calidad a la luminancia y menos a la crominancia. El valor puede ser negativo (más calidad a croma que a luma). Cuando las optimizaciones psicovisuales están activadas (**--psy-rd**) x264 resta 2 al valor introducido, ya que éstas tienden a dar más calidad a la luminancia. De manera análoga si se está codificando video en formato 4:4:4 x264 suma 6 al valor introducido, ya que al tener la croma 4 veces más resolución de lo normal se puede comprimir más sin que la calidad se resienta.

  Valor por defecto: 0

- **--aq-mode**

  Controla el modo de cuantificación adaptativa. AQ intenta redistribuir los bits asignados a un fotograma entre los distintos macrobloques de tal manera que la calidad visual subjetiva final sea la mejor. Existen 3 modos:

  - 0: AQ desactivado.
  - 1: AQ activado.
  - 2: AQ activado. Utiliza un algoritmo experimental mejorado.

  Valor por defecto:

- **--aq-strength**

  Controla la intensidad del algoritmo de AQ. Cuanto mayor sea este valor mayor cantidad de bits se destinarán a zonas con poco detalle. Nos es conveniente utilizar valores mayores de 2,0

  Valor por defecto: 1,0

- **-p, --pass**

  Selecciona el número de pasada actual. Esta opción se utiliza junto con **--bitrate**. La primera pasada genera un archivo de estadísticas con información importante para x264, y con estos datos las siguientes *n* pasadas crean una codificación más eficiente.

- **--ipratio**

  Modifica la relación de cuantificador entre fotogramas I y fotogramas P. Un valor más alto da más calidad a fotogramas tipo I comparados con los P. x264 por defecto da más calidad a fotogramas tipo I.

  Valor por defecto: 1,40

- **--pbratio**

  Modifica la relación de cuantificador entre fotogramas P y fotogramas B. Un valor más alto da más calidad a fotogramas tipo P comparados con los B. x264 por defecto da más calidad a fotogramas tipo P.

  Valor por defecto: 1,30

Parámetros relacionados con el análisis de vídeo

- **-A, --partitions**

  Controla qué particiones deben usarse para cada tipo de fotograma:

  - I: i8x8, i4x4
  - P: p8x8 (habilita p16x8/p8x16), p4x4 (habilita p8x4/p4x8)
  - B: b8x8 (habilita b16x8/b8x16)

  Cada tipo de partición aumenta la eficiencia de codificación, pero hace que la codificación sea más lenta.

  Valor por defecto: p8x8, b8x8, i8x8, i4x4

- **--me**

  Controla el método de estimación del movimiento con precisión de pixel completo. Cuanto mejor sea el algoritmo utilizado mayores probabilidades hay de

que x264 encuentre la mejor de las predicciones posibles, incrementando así la eficiencia de codificación. Sin embargo el tiempo empleado en la búsqueda es también mayor, por lo que se deberá elegir uno u otro en función de lo que se quiera conseguir. Hay 5 algoritmos, por orden de menor a mayor precisión: dia, hex, umh, esa y tesa.

Valor por defecto: hex

- **--merange**

  Controla el rango máximo, en pixeles, del algoritmo de estimación de movimiento. dia y hex están limitados a 16 pixeles, pero los demás no. Aumentar este valor aumenta las probabilidades de encontrar una mejor predicción, a costa de una bajada considerable de rendimiento.

  Valor por defecto: 16

- **-m, --subme**

  Controla el método de estimación del movimiento con precisión de menos de un pixel. El rango va desde 0 (sin precisión de menos de un pixel) hasta 11. Cada valor es más refinado que el anterior, con lo que se consigue mejor eficiencia de codificación, pero también es más lento.

  Valor por defecto: 7

- **--psy-rd**

  Controla las optimizaciones psicovisuales. En [34] se puede leer una breve descripción de dichas optimizaciones.

  Valor por defecto: 1,0:0,0

- **--no-psy**

  Desactiva las optimizaciones psicovisuales. Éstas han demostrado mejorar la calidad subjetiva de la imagen, pero afectan negativamente tanto a la medida del PSNR como a la del SSIM. Por esa razón cuando se utilizan los parámetros **--tune psnr** o **--tune ssim** esta opción se activa.

Parámetros *tune* y *preset*

- **--preset**

  Debido a la enorme cantidad y complejidad de los parámetros que controlan x264 los desarrolladores crearon los llamados *presets*, que no son más que grupos de parámetros predefinidos y seleccionados siguiendo un criterio de rendimiento. Así, se tienen los siguientes *presets*: *ultrafast*, *superfast*, *veryfast*, *faster*, *fast*, *medium*, *slow*, *slower*, *veryslow* y *placebo*. Su propio nombre ya hace intuir para que es útil cada uno, eligiendo entre velocidad de codificación y eficiencia de codificación. *ultrafast* es el más rápido con las peores opciones de codificación, mientras que *placebo* es el más lento con las mejores opciones de codificación. Como curiosidad el nombre *placebo* se lo pusieron los desarrolladores como mofa a aquellos que prefieren codificar siempre con la máxima calidad sin importarles la velocidad de codificación, ya que *placebo* comprime muy poco más que *veryslow* y sin embargo es mucho más lento.

- **--tune**

  Siguiendo el mismo razonamiento que con **--preset** se crearon los siguientes parámetros de *tune*: *film*, *animation*, *grain*, *stillimage*, *psnr*, *ssim*, *fastdecode* y *zerolatency*. Cada uno de ellos permite configurar x264 para conseguir la mejor calidad de imagen en cada situación concreta.

Otras opciones

Para acceder a la lista completa de parámetros debe usarse la siguiente línea de comandos: **x264 --fullhelp**.

REFERENCIAS

[1] Ateme, "*Why does 10-bit save bandwidth?*", URL: http://ateme.com/Why-does-10-bit-save-bandwidth [Último acceso: 30 agosto 2012].

[2] W. Gish y H. Yu, Documento JVT-H016: "Extended Sample Depth: Implementation and Characterization", de *Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6), 8th Meeting*, Ginebra, Suiza, 23-27 mayo, 2003.

[3] T. Chujoh y R. Noda, Documento VCEG-AE13: "Internal bit depth increase for coding efficiency", de *ITU - Telecommunications Standardization Sector, STUDY GROUP 16 Question 6, Video Coding Experts Group (VCEG), 31st Meeting*, Marrakech, Marruecos, 15-16 enero, 2007.

[4] International Telecommunication Union, "*ITU-T H.264 recommendation*", marzo 2010.

[5] Á. Herraiz Palomino, "*Análisis del rendimiento del codificador x264 sobre Procesador Digital de Señal*", PFC, Escuela Universitaria de Ingeniería Técnica de Telecomunicación, Universidad Politécnica de Madrid, 2009.

[6] I. E. G. Richardson, "*H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia"*, John Wiley & Sons, Ltd, 2003.

[7] E. Madejón López, "*Implementación de un sistema de codificación multivista*", PFC, Escuela Universitaria de Ingeniería Técnica de Telecomunicación, Universidad Politécnica de Madrid, 2009.

[8] http://en.wikipedia.org/wiki/File:Interframe_prediction.png [Último acceso: 30 agosto 2012].

[9] T. Wiegand, G. J. Sullivan, G. Bjøntegaard y A. Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* vol. 13, nº 7, pp. 560-576, julio 2003.

[10] Desarrolladores x264, "*Support for 9 and 10-bit encoding*", URL: http://git.videolan.org/?p=x264.git;a=commit;h=d058f37d9af8fc425fa0626695a190eb3aa032af [Último acceso: 30 agosto 2012].

[11] Z. Wang y A. C. Bovik, "Mean Squared Error: Love It or Leave It?", *IEEE signal processing magazine,* vol. enero 2009, pp. 98-117.

[12] Z. Wang, A. C. Bovik, H. Rahim Sheik y E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity", *IEEE transactions on image processing,* vol. 13, nº 4, pp. 600-612, abril 2004.

[13] I. Pulido Arranz, "*Calidad visual*", PFC, Escuela Universitaria de Ingeniería Técnica de Telecomunicación, Universidad Politécnica de Madrid, 2010.

[14] Desarrolladores libvp8, URL: http://git.chromium.org/gitweb/?p=webm/libvpx.git;f=vp8/encoder/ssim.c;hb=HEAD [Último acceso: 30 agosto 2012].

[15] Desarrolladores Subversion, "*Online home of the Apache Subversion™ software project*", URL: http://subversion.apache.org/ [Último acceso: 30 agosto 2012].

[16] Desarrolladores x264, "*x264 Main Development tree*", URL: http://git.videolan.org/?p=x264.git;a=summary [Último acceso: 30 agosto 2012].

[17] Desarrolladores x264, "*x264 web page*", URL: http://x264.nl/ [Último acceso: 30 agosto 2012].

[18] Desarrolladores x264, URL: http://git.videolan.org/?p=x264.git;a=commit;h=d9d2288cabcfd1a90f29f2f11c8cce5450a08ffa [Último acceso: 30 agosto 2012].

[19] Desarrolladores FFDShow, "*FFDshow main page*", URL: http://sourceforge.net/projects/ffdshow/ [Último acceso: 30 agosto 2012].

[20] H. Leppkes, "*LAV Filters, Open-Source DirectShow Media Splitter and Decoders*", URL: http://code.google.com/p/lavfilters/ [Último acceso: 30 agosto 2012].

[21] Desarrolladores FFmpeg, "*FFmpeg Download and Source Code Repository Access*", URL: http://ffmpeg.org/download.html [Último acceso: 30 agosto 2012].

[22] Google Code, URL: http://code.google.com/intl/es/ [Último acceso: 30 agosto 2012].

[23] Desarrolladores TortoiseSVN, "*TortoiseSVN, the coolest interface to (Sub)version control*", URL: http://tortoisesvn.net/ [Último acceso: 30 agosto 2012].

[24] P. Larbier, "*Using 10-bit AVC/H.264 Encoding with 4:2:2 for Broadcast Contribution*", URL: http://ateme.com/Why-4-2-2-10-bit-video-compression [Último acceso: 30 agosto 2012].

[25] J. Cabezas Gonzalez, "*yuv2psnrssim, Windows CLI application that, given two uncompressed video sequences, calculates PSNR and SSIM objetive quality metrics*", URL: http://code.google.com/p/yuv2psnrssim/ [Último acceso: 30 agosto 2012].

[26] Descripción del formato YUV4MPEG2, URL: http://wiki.multimedia.cx/index.php?title=YUV4MPEG2 [Último acceso: 30 agosto 2012].

[27] Internet Engineering Task Force (IETF), "*Common Format and MIME Type for Comma-Separated Values (CSV) Files*", URL: http://tools.ietf.org/html/rfc4180 [Último acceso: 30 agosto 2012].

[28] Xiph.Org Foundation, "*Xiph.org Video Test Media [derf's collection]*", URL: http://media.xiph.org/video/derf/ [Último acceso: 30 agosto 2012].

[29] Desarrolladores x264, "*Authoring a professional Blu-ray Disc with x264*", URL: http://www.x264bluray.com/ [Último acceso: 30 agosto 2012].

[30] Ateme, "*4:2:2 10-bit CM4101 H264/MPEG-4 encoder*", URL: http://www.ateme.com/CM4101 [Último acceso: 30 agosto 2012].

[31] Y. Liu, "*Analysis of Coding Tools in HEVC Test Model (HM 1.0) - Overview*", URL: http://www.h265.net/2010/11/analysis-of-coding-tools-in-hevc-test-model-hm-overview.html [Último acceso: 30 agosto 2012].

[32] A. Noé, "*Matroska File Format*", URL: http://matroska.org/files/matroska.pdf [Último acceso: 30 agosto 2012].

[33] Adobe Systems Incorporated, "*Adobe Flash Video File Format Specification*", URL: http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf

[34] J. Garrett-Glaser, "*Psychovisually optimized rate-distortion optimization*", URL: http://forum.doom9.org/showthread.php?t=138293 [Último acceso: 30 agosto 2012].